Approximate Text Generation from Non-Hierarchical Representations in a Declarative Framework

Nicolas Nicolov

Ph.D.
University of Edinburgh
1999

Declaration

I hereby declare that I composed this thesis entirely myself and that it describes my own research.

Nicolas Nicolov
Edinburgh
January 31, 1999

Abstract

This thesis is on Natural Language Generation. It describes a linguistic realisation system that translates the semantic information encoded in a conceptual graph into an English language sentence. The use of a non-hierarchically structured semantic representation (conceptual graphs) and an approximate matching between semantic structures allows us to investigate a more general version of the sentence generation problem where one is not pre-committed to a choice of the syntactically prominent elements in the initial semantics. We show clearly how the semantic structure is declaratively related to linguistically motivated syntactic representation — we use D-Tree Grammars which stem from work on Tree-Adjoining Grammars. The declarative specification of the mapping between semantics and syntax allows for different processing strategies to be exploited. A number of generation strategies have been considered: a pure top-down strategy and a chart-based generation technique which allows partially successful computations to be reused in other branches of the search space. Having a generator with increased paraphrasing power as a consequence of using non-hierarchical input and approximate matching raises the issue whether certain 'better' paraphrases can be generated before others. We investgate preference-based processing in the context of generation.

Acknowledgements

First and foremost Chris Mellish and Graeme Ritchie. In many places they kept me from saying things that I did not mean, and in as many places they helped me to say things that I did mean. As a result the thesis embodies much of their effort which I value more than I can express in words. Michael Zock who suggested the bold idea that I could come to Edinburgh and for his constant support and advice ever since. The work described here owes a great deal to many discussions I have had with him. I have tried to make his contributions clear in citations and notes, but his influence on my work is more pervasive than such devices can reveal. My PhD examiners: John Bateman and Jon Oberlander. I hope that one day I will be as thorough, exhasutive and constructive towards other PhD candidates as they were to me.

Robert Dale and Matthew Crocker who put my knowledge about computational linguistics in order. Elisabeth Engdahl for the lucid introduction to HPSG, and Mark Steedman to Categorial Grammar, Aravind Joshi for the materials he gave me on TAGs. Martin Kay who suggested the chart-based technique back in 1995, Mark Johnson who makes NLP look ever so accessible, Pieter Seuren, Ed Hovy who reviewed my research proposal. Alan Bundy whose lectures on automated reasoning represented the best course I have ever attended. There is a lot from automated reasoning that has not made its way in Computational Linguistics and his course has provided invaluable insights.

Ian Frank who if more people were like him this would have been a different world (he was my hero), Judith Good who was always smiling, Mick O'Donnell who made Systemics look not that hard after all. Saad Al-Jabri who was always helpful, Suresh Manandhar, Alistair Knott, John Levine, Jeremy Crow, Ed Carter, Steffan Corley, Kim Binstead (a very lively person). Craig Strachan, Michael Keightley, Neil Brown and Pete Spooner for, on occasions non-tivial, computing support.

The Faculty of Science and Engineering for scholarship 343 EE06006 and the Overseas Research Council for an ORS Award.

David Weir and John Carroll (I admire his efficiency), Martine Smets for interesting discussions about the DTG grammar, Olga Shaumyan who had challenging questions, Guido Minnen for the chats on constraint-based grammars. Carole Tiberius who was always there.

I also want to thank my father who initially didn't really appreciate much what I was doing (his area was number theory) until one day I solved a problem which was inherently non-deterministic by implementing a PROLOG-like search engine on top of Basic which he could run on the computers at his lab. It is a pity the bulk of his work remained unpublished. Nadia, my sister, was the person I lived for. I do hope you can go to a good university!

A lot of people have contributed to what I am and I am sure I am missing quite a few of them. It is amazing that after all this I still have so much to learn.

Contents

List of Figures

List of Tables

Chapter 1

Introduction to Natural Language Generation

This chapter sets the scene. The broad notions of natural language generation (NLG) and in particular sentence generation are introduced. This thesis focuses on techniques for sentence generation. We briefly review the overall generation process and then 'zoom-in' on the anatomy of sentence generators. We discuss the inputs and outputs of such a component, and the tasks that have to be performed.

NLG arises in the context of human-machine communication where we as humans are presented with information by an intelligent computer system. Initially, natural language generation was viewed as the opposite process of natural language understanding (NLU). In NLU the known is the text. NLU systems scan the text during which linguistic form and meaning become apparent. The NLU algorithms involve managing hypotheses concerning ambiguities at different levels. NLG is now standardly considered to include more stages than just the mapping between meaning to form. The known is taken to include the system's goals (communicative intentions). NLG algorithms involve choosing from alternatives, establishing plans, constructing specifications and realising them. The process is one of planning by progressive refinement.

> Natural language generation (NLG) is the study of how to convey an underlying message as a meaningful utterance or text using linguistic resources under numerous constraints in order to achieve communicative goals.

NLG proceeds from illocutionary intentions and perspectives (goals) to linearly ordered words. Viewed as such the process is very complex and unmanageable. Traditionally, researchers have posited different levels of representations and mappings between them in order to break the complex process into smaller sub-processes which are easier to tackle. One such decomposition of the generation task is given here below (see also Figure 1.1):

Goal identification: choosing the goals that the utterance has to achieve. These can be to impart information to the audience (persuade, inform) or prompt them to some action (ask, request). When speaking carefully and deliberately a person will try to simultaneously satisfy many goals from different sources: rhetorical, tutorial, affective and descriptive, among others.

Planning the structure of the text: This stage consists of:

- *Content determination:* detailing the conceptual information; selection (or deliberate omission) of information units to appear in the text (concepts, relations, individuals);
- *Grouping:* grouping information into sentence sized units; and
- *Sentence planning:* adoption of a coordinated rhetorical organisation of these units in a coherent text. (Some researchers consider the choice of contents words in this stage.)

Realisation of the sentence units: mapping the conceptual representations that the text planner has identified in the previous stage onto linguistic structures (also called sentence generation). Content words have to be chosen (*lexical choice*), the grammatical relations between them, closed class-words (auxiliaries, prepositions) have to be introduced (*syntactic choice*); words have to be declined (*morphological processing*).

Post-processing: generating speech (intonation and stress) or text formatting of the written output.

A two stage separation of the generation process that is often made following [Thompson 77] is the strategy/tactics distinction. A strategical component decides

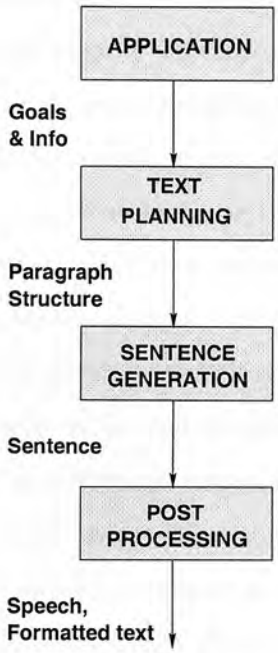


Figure 1.1: Stages in generation

on "*what to say*" and a tactical component will deal with "*how to say it*". Tactical generation is viewed as the process dealing with language specific aspects of the generation task.

The wider use of intelligent knowledge-based systems has created more areas where generation of text and speech is needed. Successful generation systems have been used in domains as diverse as: generation of explanations for expert systems; generation of weather forecasts; generation of instructions; generation of technical documentation from a knowledge base; production of summaries from tabular data; on-line information (dialogue) systems; computer-aided language learning tools; interaction with autonomous agents; etc.[1] We give a comprehensive list of many generation systems that have been used so far with brief salient characteristics in Appendix A.

[1] For overviews of the state of the art in NLG (over the years), see: [Kempen 87b, McKeown & Swartout 88, Paris *et al.* 91, Reiter & Dale 97]. There have been a number of tutorials on NLG with good expositions and overviews of recent work: [Dale 93, Zock 93, Zock 94].

In principle, the general generation architecture attempts to model cognitive behaviour (i.e., the steps that humans go through in producing language). Yet, currently, the state of the art of cognitive science and neurobiology is not such that we know for certain what the structures that humans operate with are, and what the processes that operate with these structures are. Furthermore, the hardware of computer systems differs from the human brain. Human short-term memory is limited, speech production is linear, the human brain is massively parallel. In contrast, computer systems: have larger memories, can work on non-adjacent constituents (i.e., need not consider only left-to-right production), use mostly sequential computational models. Still, given the complexity of computer systems that attempt to do a lot of the tasks in the generation model it makes sense to follow cognitive guidelines because after all we are modelling human activity (and the closer to the 'original' generators work the more natural they will seem to the humans interacting with them) and because nature has evolved strategies for achieving language production effectively (human speech production is fast given all the constraints).

In this thesis we address aspects of (what some would consider to belong to the stage of) sentence planning (how content words are chosen but not how the semantics is chunked in units realisable as sentences) and surface (linguistic) realisation (how syntactic structures are computed) from a semantic input which contains as few language specific commitments as possible. Given a particular semantic content to convey, the task is how to realise it linguistically. We look at ways of realising a semantic representation as a sentence and explore the interactions between syntax and semantics.

It is worth noting that we are concerned with text production and will not consider a whole range issues including: speech production, expressing subparts of the conceptual input non-linguistically (i.e., multimodal generation: pictures, gestures, facial expressions, etc.) and text layout.

1.1 Sentence generation

Sentence generation is the inverse problem of syntactic analysis and subsequent semantic interpretation. The latter consists in constructing some semantic representation of

an input string of words based on the syntactic and semantic rules of a formal grammar. In our work we limit ourselves to frameworks that attribute word strings with expressions in some logical formalism. The surface generation problem then consists in assigning an output word string to a semantic (conceptual) input (also often called message). In general both these mappings are many-to-many: a word string that can be mapped to several distinct logical forms is said to be ambiguous; a logical form that can be assigned several different word strings is said to have multiple paraphrases.

A sentence generator has to traverse the initial semantic structure (in some way) and make decisions about:

- **chunking**: breaking the semantic representation into parts corresponding to clauses or even smaller syntactic units (words);
- **ordering**: deriving the sequence in which elements (semantic chunks or syntactic constituents) will be considered. This refers to order of control rather than the surface order of syntactic constituents;
- **matching**: matching the semantic definitions of lexical entries against the semantic input.
- **constraining**: choosing syntactic structures for portions of the semantic input. During this process the generator is **placing** linguistic **constraints** on (possibly) partially built structures;
- **integration**: having recognised bits of the semantic input as certain linguistic constructions, these linguistic constructions have to be integrated in bigger (more specific) structures;
- **consistency checking**: if the generator adds more to the corresponding semantics than is required (by the input) it has to check that these additions are consistent with previous primary constraints (which stem from its input) or current state of affairs (knowledge about the world);
- **grammatical additions**: when the generator has exhausted the message (covered the message) and has no more semantic constraints to add to the syntactic

structure, then the current syntactic structure should be further elaborated in case it is not complete; and

- **postponing**: certain constraints/goals might not be satisfied with the generated sentence—these will be carried over to the next sentence.

From now on when we refer to generation and a generator we will have in mind the narrow meaning—sentence generation and a sentence generator. In the literature the above tasks are rarely made explicit at the same level of granularity. In fact postponing is hardly ever discussed. Chunking and ordering of the examination of subparts of the semantics are sometimes done by a recogniser for the semantic structure. In other cases it is interleaved with the syntactic chunking and ordering. One extreme is to start with a syntactic grammar and add arguments for the semantics. The idea then is to run an analyser in reverse. Rather than start with a string of words and look for a syntactic tree and a corresponding logical form, run the same analyser providing as input only the logical form. Implementations based on logic programming where such bidirectionality of the use of predicates is a common metaphor. It is true that such an approach does work on toy examples. However, a lack of appreciation of the problems which arise when attempting to scale up this approach was also the reason why a lot of NLP researchers were 'fooled' in regarding NLG as not particularly interesting. The main problems are of course that often the right variables are not instantiated and the generator has to produce (predict) blindly constituents which have nothing to do with the original semantic input. The other alternative is to augment a semantic analyser with arguments for the syntactic structure. There are couple of issues that arise immediately: semantic structures don't always look like syntactic structures (elementary semantic constituents do not directly correspond to syntactic constituents) and more importantly a separate generation grammar has to be written from scratch.

Control in generation is another dimension along which existing generators vary—on the one hand there are procedural approaches where what happens next is encoded in the current procedure; declarative approaches, on the other hand, separate the control from the data—the factual knowledge (linguistic data) is represented in a uniform format that can be interpreted by a module which contains the control knowledge.

Constraining refers to constraining the syntactic structure which opens the door for a view where the syntactic structure that is being built is only partial. Such constraints stem from the semantic structure, for example what lexical items to use in the sentence. Integration means building bigger syntactic constituents from smaller ones. This is where preferences to particular syntactic theories come into play. In some systems integration is performed implicitly by means of instantiating appropriate variables.

Constraining and integrating linguistic structures are often performed together in considering the grammar rules.

A fine level of granularity of the generation subtasks is needed in order to structure better the space of possible design decisions in the process of building a generator.

1.2 Generation as a search problem

Generation (also in the broad sense) is a structure mapping process that involves making choices. The main issue for a generation system is representing choice points and controlling multiple interacting decisions. The mapping between conceptual and linguistic structures is a highly non-deterministic task because:

- the conceptual input can be chunked (grouped into sub-parts) differently;
- a conceptual configuration can be expressed by a number of different lexical items (words);
- parts of the conceptual structure can be realised by a number of syntactic structures;
- these syntactic structures might be of different categories.

Traditionally the two important choices that are discussed for sentence generators are the choice of words and syntactic constructions:

1.2.1 Lexical choice

The problem of lexical choice is finding (open class) words that can be used in rendering the message. There are a number of factors that influence the decision:

- possible initial packaging of the conceptual input, e.g., `blond(french(woman))` is likely to be realised as *blond French woman* but not as *French blonde*;
- selectional restrictions: *essen/fressen*;
- collocations: *tall person/high mountain*;
- formality: *Hi!/Good morning!* ;
- salience: *borrow/lend*
- user model of the lexicon of the user: *modem/connecting device*;
- syntactic constraints: *The pen belongs to John\ * is had by John.*

1.2.2 Syntactic choice

The choice of syntactic constructions involves decisions about the shape of the linguistic structures. This depends on:

- possible previous priming (i.e., bias to a particular syntactic construction, e.g., in dialogue or question answering the answer often preserves the structure of the question);
- what words will be used: *I am considering buying a computer/I want to buy a computer*;
- topic/focus: topicalisations (*John loves Mary/Mary John loves*).

Lexical and syntactic decisions are interdependent. For example the verb *shipwreck* can only be used in passive:

They were shipwrecked off the coast of Newfoundland.

Different approaches position the lexical and syntactic choices differently. Some consider lexical choice to be part of the sentence planning. In fact the majority of systems perform lexical choice before syntactic choice in a pipeline fashion. An exception is Systemic Grammar where lexical choice happens after syntactic decisions have been made.

1.3 Non-hierarchical semantics

In this section we take a closer look at the nature of the input for NLG systems and argue that it should not have certain hierarchical dependences. We study this issue from a multilingual as well as monoligual perspective.

1.3.1 Tree-like semantic assumption

Early work on sentence generation assumed input of the form: $\texttt{pred}(\texttt{arg}_1, \ldots \texttt{arg}_n)$ and the generation process was reduced to mapping $\texttt{pred} \rightarrow$ *verb*, $\texttt{arg}_1 \rightarrow$ first *complement*, etc. This approach, of course, makes the "semantic structures" be nothing more than disguised syntactic representations and reduces the sentence generation problem to finding out the ordering of the constituents. This 'tree-like' semantic assumption severely restricts the paraphrasing power of generators and (among other things) does not allow for handling translation mismatches (head switching examples) and incorporation of modifiers in the syntactic head.

1.3.2 Multilingual generation and machine translation

Generation in a number of languages and generation in the context of machine translation (MT) raises important questions about the nature of the (conceptual) input. Standardly, the message is cast in terms of a logical form or as an expression in some knowledge representation language. In our work we consider an interlingual approach (which has advantages with regards to modularity) and expect the input to be able to make distinctions that will motivate the choice of one linguistic structure over another in all languages.[2] This means that the conceptual ontologies that give structure to the set of concepts (building blocks) in the message will have to be richer.

Translation mismatches. Various kinds of non-perfect alignment of direct equivalents[3] (referred to as translation mismatches) have been considered in MT (for a survey

[2] In contrast transfer-based approaches to MT use language dependent 'semantic' representations that express the distinctions relevant for the target language only.

[3] And sometimes non-existence of direct equivalents.

see [Dorr 93, Dorr 94]). Some of these mismatches can be handled straightforwardly in the generation context. There is one case, however, that of head switching, which is hard to reconcile with the tree-like semantic assumption.

Heads[4] of phrases incorporate part of the semantics of the initial (input) semantic structure and to a large extent predetermine the final syntactic structure. Which bits of the semantic input should be grouped together (packaged) into the lexical head of the phrase is a decision to be taken by the generation component and not by another module. Certain groupings of semantic features are only meaningful for a language if in the language there is a potential to express them with a word. Deciding on what the right head should be is challenging, because one cannot assume that the semantics contains this information (the answer depends on the language one is generating in). A bad solution to the head selection problem could lead to a dead end.

The phenomenon when the target language head of phrase has to be different (because of lack of similar lexical material or syntactic constraints) is termed *head switching*. In cases of head switching a semantic configuration of process, participants and circumstances is realised with 'radically' different surface structures. A famous example is the incorporation of the manner information into the head verb for English and its separate realisation as a modifier in Romance languages:

English: *He swam across the river.*
French: *Il a traversé la rivière à la nage.*

In effect the (interlingual) conceptual input can be chunked in different ways giving rise to different linguistic structures.[5] Here is another example:

English: *He almost fell.*
French: *Il a failli tomber.* [Janes 88]

Head switching is a serious problem for MT systems where the source and the target

[4] Every phrase has a subcomponent that largely predetermines the combinatorial properties of the whole phrase. These are called head daughters (or lexical heads when the subphrase is a single word). Thus, in the sentence *The computer sent the message*, the head daughter is the phrase *sent the message*. The verb *sent* in turn is the head of the verb phrase and it is the lexical head of both the verb phrase and the sentence [Pollard & Sag 87].

[5] In contrast transfer-based approaches to MT use language dependent 'semantic' representations that express the distinctions relevant for the target language only.

languages exhibit different flexibility in their capacity to incorporate semantic elements into words.

1.3.3 Monolingual arguments

While primarily a concern of MT the problem of head selection (packaging information and determining syntactic structure) persists in generation as well, because certain paraphrases in the target language also show that information can be put together in different ways:

Fred limped quickly across the road.
Fred hurried/rushed across the road with a limp.
Fred crossed the road limping quickly.
Fred, who was injured in the leg, crossed the road quickly.
The lame Fred crossed the road quickly.
The cripple crossed the road quickly.

Consider also the pair:

She smiled a welcome to the guests.
She welcomed the guests with a smile.

The reason why head switching cases are problematic for MT stems from the fact that often the semantic representation for the translation equivalents contains commitments which might be natural for one of the languages but not for the other. Often such commitments come down to dominance relationships (for example in logical/PROLOG terms, feature structures, etc.) which by virtue of the particular KR formalism which is used have to be stated. That is why the ability to express the semantics in as abstract manner as possible becomes of importance.

Flexible modifiers. Frameworks that assume hierarchical (tree-like) nature of the input cannot:

1. produce different ordering of modifiers. The order of modifiers reflects the order in which they appear in the logical form (we use rewrite systems notation[6]):

[6] Indeed generation can be seen as an instance of a rewrite system [Book & Otto 93, Baader & Nipkow 98] — the semantic representation is rewritten as a syntactic representation.

`mod1` $\rightarrow$ *adj1*	
`mod2` $\rightarrow$ *adj2*	
`concept` $\rightarrow$ *noun*	
`mod1(mod2( concept ))` $\not\rightarrow$ *adj2 adj1 noun*	

2. incorporate modifiers in order other than in the order in which they 'wrap' the semantics is not possible, i.e., such approaches cannot simultaneously generate *French blond* and *blond French girl* and *blond French*:

`french`	$\rightarrow$	*French*
`blond`	$\rightarrow$	*blond*
`girl`	$\rightarrow$	*girl*
`french(blond(girl))`	$\rightarrow$	*French blond*
	$\rightarrow$	*French blond girl*
	$\not\rightarrow$	*blond French*

If one accepts the idea that the underlying semantics for the above cases is the same natural question that arise are: 'How is the semantics represented?' 'What guides the different restructuring/packaging for different languages?'. 'How is the head realised lexically?' and 'What syntactic structure is used?'.

In order to address the above issues (head switching and flexible modifiers) we argue that a more abstract, non-hierarchical representation for encoding the conceptual input is needed. We have shown that naive generation approaches can't do the right thing with tree-shaped semantics. If the NLG system is going to be based on some kind of simple compositional pattern-matching on the semantics then the semantics has to allow all sorts of possible trees to be imposed on it. Hence, it has to be a more general kind of structure. Generators then will have more flexible ways of matching linguistic resources (such as lexical items and the semantic structures associated with grammar rules) against the conceptual input.[7]

1.4 Approximate generation

There are MT translation pairs for which language motivated semantic representations which aren't really the same and none of them subsumes the other (at the language

[7] Similar arguments have been made in the context of semantic transfer [Dorna *et al.* 98]. Dorna et al. show how by using a nonhierarchical representation the head switching is no longer manisfest in the representation.

level these involve lack of direct equivalents). See [Barnett *et al.* 94] for a discussions of examples. In order for us to allow for common input we need to use finer grain representation (as mentioned above) and also be able to consume and match partially the input. Allowing approximate matches raises the question whether in generation we should consider linguistic structures that express the whole conceptual input (completeness) *and* only the conceptual input (coherence).

> "Translation must often be a matter of *approximating* the meaning of a source language text rather than findingan exact counterpart in the target language since languages differ in the concepts and real-world entitirs for which they have words and grammatical constructs." [Kameyama *et al.* 91, page 197] [emphasis added]

One important underlying assumption of this research is that language can feedback to the planning component, i.e., humans do say things they had not originally planned because a particular language construction forces them to include additional semantic material. Detransitivised verbs in one language might not have a counterpart in another and thus the latter language might need to introduce additional material. A similar example involves definiteness and number information which are required in English but are mostly lacking in Japanese, whereas the honorificity and speaker's perspectivity information is required in Japanese but mostly lacking in English. There are fundamental discrepancies in the extent and types of information that the grammars of these languages choose to encode. Consider the following pair [Kameyama *et al.* 91]:

English: *I like Matisse's drawings better than paintings.*
Japanese: *watasi wa Matisse no aburae ya suisaiga yorimo senbyou ga suki desu*

Japanese doesn't have counterparts for drawings or paintings and needs to approximate them using (in this case) more specific concepts: *I like Matisse's line_drawings better than oil_paintings or water_colours.*

Current generation systems only worry about how much of the original semantics has been consumed/covered and their termination condition is that the current derivation must cover all of the input semantics. This model is good for isolated cases where one is sure that the input semantics is expressible as a single sentence. Yet, this model is rather simplistic in a number of respects. Not only do humans introduce additional

semantic material, but it may happen that a certain concept (or a subgraph in the general case) is conveyed by an expression whose lexical semantics is a concept more general or more specific than the original concept in the input. This is the reason why we want generators to keep track of the 'shadow' semantics corresponding to the current derivation. In order to avoid serious deviations from the conceptual input we consider that the shadow semantics is constrained between an upper and a lower semantic bound (which we take to be part of the input specifications).

1.4.1 Our notion of paraphrase

We have a more liberal notion of what we take to be synonymous sentènces (because we allow for approximate matching and do not want to guarantee exhaustive consumption of the conceptual input). This relaxes the implicit strict assumption of other generation systems that the input is realisable as a single sentence. In normal situations people never speak strictly: potential semantic contrasts are often ignored in actual contexts as irrelevant to the main point of the message. This ability of the speaker not to notice semantic nuances if he does not need them is an important property of natural languages that has not been accounted for.[8] A similar view is shared by Mel'cuk:

> "Contextual neutralisation of semantic differences is extremely important to any semantically-based theory of language ..." [Mel'cuk 88, p.86, remark 4]

One doesn't really need to go beyond the sentence level to start noticing these effects. Consider for example the translation pair [Barnett *et al.* 94, p.358]:

Spanish: *Vi una perra con sus cachorros.*
English: *I saw a dog with her puppies.*

The Spanish noun *perra* refers to a female dog, but the pronoun *sus* does not indicate gender of the antecendent. In English, the information is the other way around: *her* indicates gender, while *dog* doesn't. The Spanish morphological information gets absorbed by a syntactically remote part of the English sentence. It could be suppressed altogether by using *its* in place of *her*.

[8] We introduce the exact notion of a paraphrase in Chapter 5 where it will be defined as a sentence with corresponding semantics within certain upper and lower bounds.

While it is in principle feasible, the thesis is not in fact going to consider aspects of contextual neutralisation related to discourse, and is focussed instead on sentence realisation.

1.5 The problem of logical form equivalence

In previous sections we worried about the form of the input. There is a negative theoretical result which amounts to exact generation being NP-hard.

Generators normally use a grammar (in the broad sense, i.e., not just a syntactic grammar) which associates a single logical form with each possible meaning of a sentence. Shieber notes that there are infinitely many intentionally equivalent logical forms (i.e., that mean the same) yet the grammar only relates a very small number of these to linguistic forms in the language [Shieber 93]. Such prominent logical forms are called canonical forms.

There is a problem if the generator is given a non-cannonical logical form to express. It should look for a cannonical form equivalent to its input but for representation languages that have the power of first order logic logical equivalence is in general not computable.

One could try to look for restrected logics whose equivalence problem is computable but which still represent all NL meaning distinctions (the "AI problem").

For restricted domains, one can impose well-formedness constraints based on domain dependent principle which restrict the granularity of meaning distinctions.

If the generator is always given a cannonical form, this implies that the input isn't really a logical form but an 'abstract syntactic representation'. Thus, language dependence is built-in in the non-linguistic components of a full generator and modularity is lost.

1.6 Non-deterministic generation

Generation approaches can be classified into two main groups: deterministic generators which produce a single sentence for a given input and non-deterministic generators which can come up with a number of paraphrases for a single semantic input.

Deterministic generators are fast because they are assumed always to make the right decision and need not store information about alternatives at different points in the generation process so that they can return to these choice points later on. In order to produce different sentences there must be differences in the input that would motivate one structure rather than another. Thus, deterministic generators need to make finer distinctions in the input in order to produce different sentences. Systems based on this model include: PENMAN, COMMUNAL, KOMET, KPML (based on Systemic Functional Grammar) and IDAS (which uses classification).

There is a strong psycholinguistic argument against determinism in that people do not always produce the same sentences in the 'same' situations. People also produce ungrammatical sentences (which deterministic generators are not geared to) and often they need to 'repair' what they have said. In addition determinism imposes very strong constraints on the input to the generators—it has to be very fine-grained.[9] In building computational systems we are not obliged to model human behaviour[10] yet there are arguments why it is worth considering human behaviour namely:

1. Communication is at the heart of social interaction and verbally expressing oneself is a major part of communication. Human language capacities have evolved over a long period of time.

2. Humans generate language in real time although the search space is large while the size of the short term memory buffer is small (7 ± 2 items).

---

[9] Which cannot always be guranteed especially if the input comes from non-linguistic applications.

[10] Just as when building airplanes designers are not trying to imitate the dynamics of the wing movement in birds.

Non-deterministic generators can render a given input differently. At each point of the generation there might be a number of alternatives which can be pursued. The generators will need to remember these alternatives (or more precisely the alternatives that remain to be explored) so that they can come back to them in case something goes wrong further down the path they will choose. This returning to old choice points is called backtracking and is typical for a lot of AI problems involving complex search spaces. A number of generators rely on backtracking: FUF, SHDG, PROTECTOR. From a psycholinguistic point of view backtracking[11] can be seen (somewhat) as the analogue to repairs that humans make.[12] Furthermore, there are situations when paraphrasing capabilities are required:

1. In dialogue systems paraphrasing is an important issue ("*Could you say that again?*" situations).

2. If the output of the generator is post-evaluated and if the post-evaluator can reject the output. It can be argued that such post-generation constraints can be integrated within the generation process but then modularity can be lost.

3. In generating paragraph-length texts often it is useful to avoid repetitive constructions. Unless generators have preferences to using syntactic or stylistic constructions that can be changed dymanically generators will be producing the same set of paraphrases in the same order. Thus similar semantic structures will be expressed using similar syntactic constructions. If variation is needed then in consequent generation of sentences the first solutions might have to be ignored.

4. If the generator is allowed to fail (i.e., reach a dead end—neither of the available alternatives is consistent with the choices made so far).[13]

The last point is quite alarming as it means that a generator might not produce any output at all. We now turn to the notion of failures:

[11] We would like to stress that the non-deterministic aspect of generators is not the same as the ease with which non-deterministic algorithms can be coded in the particular implementation language that is used.

[12] Humans also explore the search space and in encoutering dead-ends opts for local repairs. This allows for speed of processing.

[13] This requires paraphrasing capabilities only if the failure is local and there is a successful result elsewhere in the search space.

gaps: 1. syntactic: inability to express a piece of semantics as a certain syntactic category.

Languages in general are very flexible so often the cause for such failures can be due to the lack of appropriate mapping rules (i.e., given that languages are flexible there are less likely to be gaps. Thus the gap is not in the language but in our inexhaustive encoding of the linguistic structures).

2. lexical: lack of a word in the language expressing the given concept. One can resort to definitional capabilities (expanding the concept using simpler concepts). That often leads to a non-global optimum.

semantic failures: the entire semantics has not been expressed (very common with syntactic grammar-driven generation) or (too much) additional information has been added, i.e., there is a mismatch (beyond a certain level of acceptability) between the input semantics and the semantics of the generated sentence. Generators that are not coherent and complete are particularly subject to this property.

syntactic failures: the generated syntactic structure does not satisfy some global constraints.

stylistic failures: the generated structure might not satisfy certain stylistic constraints: sentence is too long, it can lead to false unwanted implicatures, can be ambiguous, might not be at the right level of politeness, etc.

The gap failures are local and the others are global. The global failures are of the post-evaluation kind, i.e., they are best detected after the generator has produced some result. Local failures force the generator to consider another set of choices. Usually this is done by backtracking to the previous choice point and taking an alternative route.

Re-generation vs backtracking. Even if a generator has to produce only one sentence (at the macro level) it does not mean that the generator should be deterministic (i.e., not use backtracking). Backtracking is obviously needed if more solutions/paraphrases are required but *best-first* (or preference-based) performance approaches do not necessarily imply a deterministic generation architecture.

Lexical gaps. Later in the thesis (in Section 6.1.1) we consider the generation of the following sentence:

Alexander launched a full-scale attack on the town.

Following a sensible *top-down* generation strategy the generator will get stuck.[14] The reason will turn to be that we will not be able to realise a certain concept given a goal syntactic category. Such is the nature of lexical gaps: The absence of a word in the lexical field of a language is called a lexical gap.

A key feature of lexical gaps is that they are idiosyncratic and language-dependent. One could not expect a language-independent input to specify enough information to enable them to be avoided. Thus, lexical gaps constitute a serious problem for top-down deterministic generators or generators that have imposed on them prior lexical decisions.

1.7 Declarativeness

So far we have been stressing that analysis and generation are very different, they have different sets of concerns. Parsing is concerned with structural alternations while generation is about functional choices. However, both parsing and generation processes establish the (grammar) relation $Meaning \leftrightarrow Form$, i.e., they are the two *performance* sides of linguistic *competence*. From an engineering perspective it makes a lot of sense to use a single declaratively stated relation between meaning and form. The fact that for effieciency reasons we might use different indexing (on the linguistic structure for analysis and on functional and semantic information for generation) should not come as a big surprise. It is true that the largest generation and analysis systems have not been built to allow reusability of the linguistic resources for the other process.

In this thesis we take the linguistic resources that the generator uses to be declarative in nature and consider techniques for efficient processing with these. We think that even from the point of view of generation alone it is important to use declarative data because it is useful to consider different generation strategies (presently it is not

[14] By the time we reach Section 6.1.1 we will have more machinery and associated notation to illustrate the example more concretely.

known what strategy is the most efficient nor which one is the most psychologically plausible). If a lot of effort is spent developing a large grammar that can be used in one mode of processing and later arguments are found that the particular strategy is not good (doesn't scale up, cannot be integrated with other processes, cannot take other knowledge sources into account, is not psycholinguistically plausible, etc.) all this development effort will be wasted. Furthermore even if we want to use a single generation strategy for the purposes of maintaining the grammar it is better if control is separated from the data (issues like these have been extensively discussed in the logic programming literature [Hogger 91]).

1.8 Contributions of this thesis

Current generation systems exhibit a fair amount of sophistication and many are successfully used in various domains. In order for us to build even better systems (systems that will allow for more natural communication with humans), current theories, architectures and implementations need to be improved. Here are some enhancements over the current state of the art generators which are addressed in this thesis (the main contributions follow this list):

- Planners cannot be expected to know about what a sentence is (size). Languages are flexible and can express a lot of information in one sentence.
- Sentence generators should not have lexical choices imposed on them. If they do the search space is severely restricted (sometimes to the point where there is no solution). In a way the generator then can be reduced to a parser (word adjacency is swapped for semantic connectivity or embeddedness).
- The conceptual input should not be prepackaged because it then becomes an abstract syntactic representation and paraphrases are ruled out. This severely hinders multilinguality.
- Sentence generators should not be deterministic (either at the macro or micro level). It is not always possible to make an optimal choice.
- Generators should not duplicate work.

- Generators cannot be expected to express a perfect match of the conceptual input.
- Generators cannot be expected to have syntactic decisions imposed on them because there are paraphrases which have radically different syntactic structures.
- Generators should not be procedural. It is not known what the best strategy is. The resources should be separated from the processing for better maintenance.
- Generators cannot be expected to come up with the best paraphrase. They need additional mechanisms.

The main contributions of this thesis are:

1. use of non-hierarchical conceptual input;
2. use of approximate matching between the input semantics and the semantics corresponding to the generated sentence;
3. use of a linguistic framework which improves on one of the widely used grammars for generation;
4. use of memoization techniques to keep the results of previous generated subphrases and reuse them in subsequent goals;
5. developing a preference-based model for generation;
6. developing a practical generation system which embodies the above ideas.

We have formalised and implemented an approach to NLG that has the above properties.

1.9 Structure of the thesis

Chapter 2 puts our work in the context of the existing research in sentence generation. A number of **approaches** are surveyed thus illustrating the variety of assumptions and design decisions adopted in generation work so far. More importantly, the weak and strong points of the frameworks are pointed out both from

a computational and linguistic point of view. Stand-alone (dedicated) generation systems are reviewed; attention is also paid to machine translation systems which face more problems in cases of mismatches between the source and target languages. It is noted that existing generation approaches often assume a hierarchical structure on the input semantic representation which is not justified independently of the language. Furthermore, the constraints on the semantics of the output (completeness and coherence) are more restrictive than necessary and intuitively good realisations are ruled out.

Chapter 3 introduces the formalism that is used for representing the **input** for the system (conceptual graphs). The conceptual graphs formalism allows us to describe the input semantic representation without imposing a hierarchical structure on it which is biased towards a particular language.

Chapter 4 discusses the grammatical framework which is used for expressing the syntactic structures that the system **outputs**. We consider a particular class of grammatical formalisms — non-concatenative grammars. We look in particular at Tree-Adjoining Grammars (a formalism which has been used much in generation) and a later development based on them — D-Tree Grammars. The use of D-Tree Grammar as will be shown in Chapter 5 simplifies the mapping between semantics and syntax.

Chapter 5 contains the main contribution — it describes the **generation model**. Firstly, the knowledge sources are introduced. The grammar of the system is a set of mapping rules which relate semantic representations cast in terms of conceptual graphs to syntactic structures stated using syntactic trees in the D-Tree Grammar framework. The declarative aspect of the mapping rules is emphasised which has important ramifications for the processing model — namely that different strategies for generation can be exploited. Then the actual generation model is described. It is also illustrated with concrete examples. Then the notion of a paraphrase is defined. A valid paraphrase is not only a single sentence but can also consist of an initial sentence and a possible follow-up. An important characteristic of the generation framework is that the semantics of a paraphrase is not taken to be just more specific (completeness) or more general (coherence)

or both but has to match the input semantics. This matching is meant to allow for cases that previous systems will rule out where there could be combinations of specialisation and generalisation of parts of the input semantics. Ways for comparing alternative paraphrases are discussed. This relaxed notion of a paraphrase will in general allow many more possibilities — the search space will be larger. To address this, mechanisms to control the search are investigated.

Chapter 6 motivates the use of **memoization techniques** through an example that shows that standard backtracking algorithms for generation will need to duplicate a lot of work. We summarise chart parsing and review existing approaches to chart generation. We then define chart generation from a semantic perspective. As in parsing, there is an array of different processing strategies that can be explored (a top-down chart strategy is implemented).

Chapter 7 discusses a model of **preference-based generation** which aims at producing better sentences earlier. We talk about different dimensions along which valid paraphrases can be compared. A method for comparing semantic structures is suggested which allows the generator to rate the paraphrases on semantic grounds. Similar preferences can be defined for other aspects of the resulting structure (the syntax for example) and the preferences can be integrated dynamically in the generation process. We investigate a strategy which uses an agenda of unprocessed goals which is implemented as a priority queue so that a best-first search can be performed.

Chapter 8 concludes by summarising the contributions of this work. By combining two powerful frameworks — that of Conceptual Graphs and D-Tree Grammar, a generation framework has been developed which allows for the linguistic realisation problem to be viewed in a more general setting. Important starting assumptions have been relaxed: (1) that of the hierarchical nature of the input semantics, and (2) inflexibility in covering the conceptual input. From the algorithmic point of view we have considered (3) memoization techniques, and (4) preference-based processing. The realisation component is viewed in a broader sense addressing lexical choice and the generation of follow-up sentences. The generation system that is built is viewed as a general test-bed for experimenting

with possible generation strategies. Taking additional knowledge sources into account during generation is also discussed. The major limitations are highlighted and directions for future work are outlined.

A comprehensive list of generation systems is given in an appendix which is followed by an index.

SUMMARY

- ⊙ NLG is concerned with the production of text by a computer.
- ⊙ Sentence generation is at the heart of a NLG system.
- ⊙ Generation (in the broad and narrow sense) is a goal-oriented process, a process of choice.
- ⊙ Generators should not assume a tight relationship between semantics and syntax.
- ⊙ Generators cannot always make the right decisions.
- ⊙ In realistic applications it is not always possible to express all the input and only the input (and at the given level of specificity).

Chapter 2

Literature Survey

In this chapter we study existing approaches to sentence generation. Our goal is to examine to what extent the questions we have raised at the end of Chapter 1 have been addressed in previous research. We first motivate the need for linguistically motivated syntactic representations in generation by going from a simplistic approaches (direct mapping and templates), observing some deficiencies, and moving on to current state-of-the-art approaches. We describe generation as decision making (also known as grammar-driven), message-driven methods, generation using unification, classification and incremental consumption methods from non-hierarchical representations. One of the issues we are concerned in the thesis is declarativeness. We even show a radically different procedural[1] approach taking hirarchical input, yet still attemping head switching examples.

2.1 Introduction

For sentence generation the output that one is interested in is of course a sentence (a string of words). We start this chapter with a discussion of two of the simplest ways of doing generation (direct mapping/replacement and template-based generation).

[1] Very often in generation work, the PENMAN system [Mann 83] is given as the classical example of a procedural approach to realisation. This is of course true as regards this early implementation of Systemic Functional Grammar [Halliday 94] is concerned but the framework can be considered in a more declarative setting.

2.1.1 Direct mapping

A simple way for a generator to come up with a string of words for a given meaning is to have the generator keep a collection of meanings and their realisations. Generation then is a table look up process. There are many examples from computer software. Systems enter a certain state and want to communicate to a non-expert what the problem is:

```
LaTeX Warning: There were undefined references.
(see the transcript file for additional information)
```

Associating meanings directly with ready made strings of words has a number of disadvantages:

1. All meanings that the generator can accept have to have been anticipated and put in the table in advance. The approach doesn't scale up.

2. It views the semantics as an atomic entity with no internal structure. However, some parts of certain meanings are expressed in the same or similar way, and there are no mechanisms for this to be stated. This approach does not capture generalisations at all.

2.1.2 Templates

Given that certain meanings have similar realisations except for some parts of the string, it is interesting to consider whether the semantics cannot be arranged in such a way so as to separate the "constant" semantic parts from those semantic parts which can be realised differently. At a first approximation the generator can map the input meaning to a string some parts of which are unknown (those are the parts that correspond to the parts of the meaning which can vary). Then the system can go and explore how these different subparts of the meaning can be realised. This can be seen as a sort of fill-in-the-gaps [Reiter 95].

A specification what meaning structures can be mapped onto what partial strings (strings with holes in them) is called a template.

Here is a simple example:

```
$ rm garbage
garbage:  No such file or directory
$
```

One attempts to remove a non-existent file which generates a 'file-not-found' scenario and the system uses a template "X: `No such file or directory`" and substitutes the name of the file for X.

Templates can be embedded in the sense that in order for the generator to explore how a certain subpart of the meaning is realised it looks again at the sets of templates to find an appropriate one. The generation strategies for templates are top-down. The final string is the result.

All manipulation of non-syntactic/semantic template systems is done at the character string level. Template systems do not attempt to represent the text in any deeper way (in particular no syntactic structures (trees or dependency graphs) are produced).

Template-based systems can be difficult to modify according to changing user needs. Making even a slight change to the output of a template-based generator may require a large amount of recoding (of programs) and rewriting of templates. In unforeseen circumstances such systems can generate wrong, ungrammatical text or one which is misleading, i.e., has false implicatures.

2.2 Generation as decision making

Perhaps one of the most widely used approaches to sentence generation and generation as a whole views generation as making functionally motivated choices. This approach is based on Systemic Functional Grammar (SFG) which was developed by Michael Halliday in England in the 1960s [Halliday 94] which in turn grew out of anthropological [Malinowski 23] and sociological linguistics [Firth 57], as well as the Prague School of Functional linguistics. Rather than describing the internal constraints on valid syntactic structure (syntagmatic dimension) SFG concentrates on how certain (minimal) alterations in the language are motivated by the communicative goals (paradigmatic dimension). Thus language is viewed as a resource for achieving aims (cf. unbounded

resource planning in AI). This perspective has proved very useful in NLG and one of the early large generation systems PENMAN used exactly this approach. Surface forms are viewed as consequences of selecting abstract functional features (in the process of making choices) and are gradually refined. SFG takes into account a wide range of input constraints structured in multiple descriptive dimensions (called metafunctions):

ideational propositional meaning of a clause (captured in the transitivity system of a clause);

interpersonal why sentences are uttered (expressed in the mood structure);

textual the glue that holds communication together, based on information packaging needs.

The grammar (meaning-form relation) is organised as a collection of choice systems representing simultaneous labeled alternatives (exclusive disjunctions) corresponding to linguistic alternation (i.e., the structure of the system network is language dependent). Each system has an entry condition. Organising generation in terms of functional choices very quickly restricts the search space of available forms of expression.

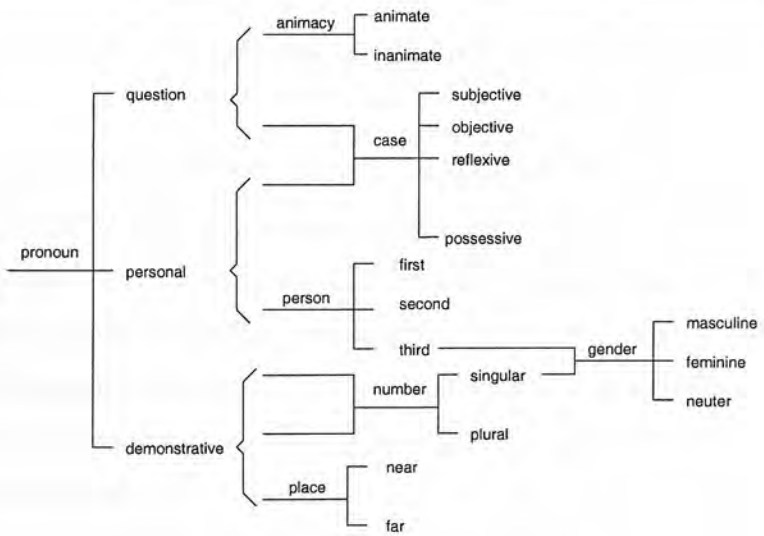


Figure 2.1: System network

Figure 2.1 taken from [Mellish 91] shows an example system network. The selection of one alternative determines what other choices are available (consequent systems

are called more delicate). An opening square bracket represents alternative exclusive choices available at that level (we will see later how the system networks can be used, i.e., how one gets to a certain system). Opening curly bracket represents a number of choices regarding each of which a decision must be made. A closing square bracket indicates that the traversal on the right is reached if any of the options on the left are reached. A closing curly bracket represents a system which can be traversed only if all of the options on the left have been reached.

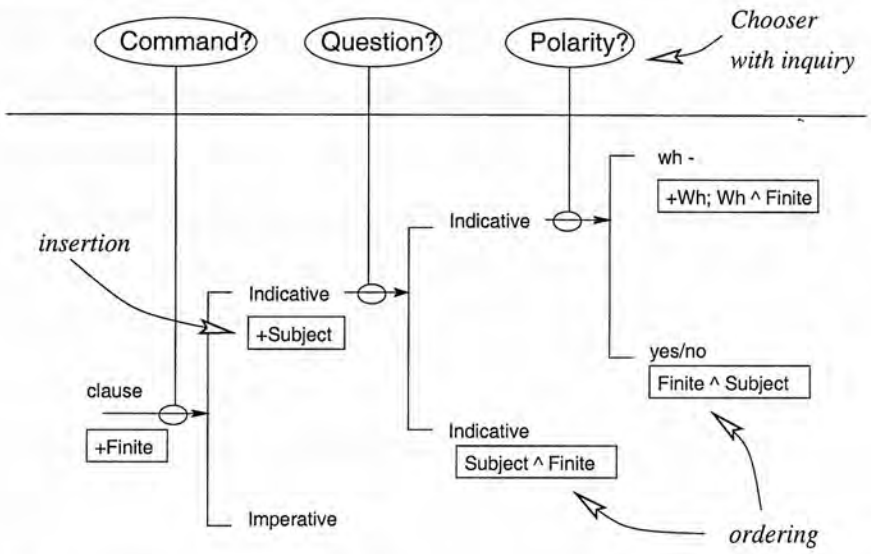


Figure 2.2: Realisation statements and inquiry semantics [Matthiessen & Bateman 91]

A system network does not build linguistic structures. This is done by realisation statements associated with (functional) features in the system network. Realisation statements specify a structural fragment (syntagmatic organisation) if a certain feature (from the paradigmatic dimension) has been specified. Structure is gradually built up in terms of functionally labeled constituents that are first introduced and then further constrained by realisation statements. Figure 2.2 shows a system network augmented with realisation statements.

We haven't yet explained how traversing happens and how the semantics is accounted for. This is done through *choosers* which are procedures associated with each system which choose between the system's features by asking one or more questions (*inquiries*). Choosers and their inquiries form the semantic interface. Figure 2.2 also shows the

choosers (in ovals) and vertical lines representing the association with systems in the system network. Choosers are organised as decision trees.

A systemic grammar can be used in the following way:

1. start with rank of least delicacy
2. make choices until maximally delicate distinctions offered have been drawn, i.e., traverse relevant system networks:
   - result is a complete description of a linguistic unit at that rank (the features for the whole phrase are chosen/inferred).
   - this set of features is called the selection expression.
   - Realisation statements associated with each feature in the selection expression are applied. Thus, constituents (functions) of the phrase become defined. The selection expression will classify a linguistic unit in terms of all metafunctions.
3. The process is applied recursively for each constituent (defined function) of the phrase.

Thus the main components of the generation algorithm are:

1. traversal algorithm for walking through the system network;
2. structure building algorithm for applying realisation statements;
3. chooser interpreter for traversing the decision trees of the choosers.

There is of course a lot more to be said about generation within the SFG framework. Good descriptions of SFG generation can be found in [Davey 78, Mann 83, Mann & Matthiessen 85, Patten 88, Matthiessen & Bateman 91, O'Donnell 96].
There have been a large number of NLG systems based on SG: PROTEUS [Davey 72, Davey 78], PENMAN [Mann 83, Mann & Matthiessen 85], SLANG [Patten 88], IMAGENE [Linden *et al.* 92], COMMUNAL [Fawcett & Tucker 90], KOMET [Bateman *et al.* 91], KPML [Bateman 96], WAG [O'Donnell 96], GENESYS [Fawcett & Tucker 90], MULTEX, TECHDOC [Rösner & Stede 94], GIST [Consortium 96], DRAFTER [Paris & Scott 95].

At the onset of our research we didn't choose to work with PENMAN because of the procedural nature of the chooser inquiry semantics. Subsequent implementations use declarative alternatives.

Furthermore, PENMAN performs lexical choice at the end (most delicate systems) and is thus likely to reach a dead-end in case of lexical gaps.

Systemic generation is deterministic. The input is expected to provide enough information so that the systems can be navigated until they are entirely traversed. Some computational implementations provide defaults for underspecified inputs. Alternatively an underspecified input could be (non-deterministically) expanded as fully instantiated (ground) representations from which a deterministic generator could produce a number of paraphrases. Deterministic generation impies that decisions must be specified by the system developer in the order in which they will be executed.

Systemic generators work top-down. Elke Teich notes a problem in top-down generation, i.e., that the choice of gender at the NP level which happens before the head noun is chosen. If the gender is grammatical the decision at the NP level is arbitrary. This stems from a combination of factors, including not having a means for sisters to restrict each other in systemics, e.g., feature-sharing or daughter dependency.

In the case of PENMAN complete coverage of the conceptual input cannot be ensured because it is consulted only when needed by the choosers. Thus, if the input contains elements for which there is no chooser that will ask anything about them they will pass completely unnoticed.

The SFG approach to generation is often referred to as grammar-driven generation because we are looking at the available resources in the language and only if there are distinctions made by the language do we then consult the semantics.

2.3 Message-driven generation

Grammar-driven generation is often contrasted with message-driven generation which might seem a more natural way of viewing generation given that we would want to do indexing on the semantic structures in the specification of the Meaning↔Form relation. In message-driven generation the emphasis is on the message (and its parts) and for a certain configuration the interesting questions is to determine the alternative linguistic forms for expressing the conceptual configuration and what are the constraints on their use.

An example of a message-driven generator is the MUMBLE system [McDonald 81, McDonald & Pustejovsky 85, Meteer *et al.* 87]. MUMBLE was built in order to provide a psycholinguistically plausible model of speech production—it produces word streams in a strict left-to-right fashion. MUMBLE emphasises the structural view of language and it uses a constituent structure for the linguistic forms. MUMBLE uses Tree-Adjoining Grammar. The system assumes that lexical choice has been performed before linguistic realisation. Also the way a modifier is attached to the syntactic head must be specified as the value of the `attachment-function` slot (see Figure 2.3).

```
(( discourse-unit <r1>
    :head (general-clause <r2>
            :head (give <r3>
                    (general-np <r4>
                      :head (np-proper-name "John") <r5>
                      :accessories (:number singular
                                    :determiner-policy no-determiner))
                    (general-np <r6>
                      :head (np-proper-name "Mary") <r7>
                      :accessories (:number singular
                                    :determiner-policy no-determiner))
                    (general-np <r8>
                      :head (np-common noun "book") <r9>
                      :accessories (:number singular
                                    :determiner-policy kind))
                      :further specifications
                        ((:specification
                          (predication_to-be *self* (adjective "blue"))
                          :attachment-function restrictive-modifier)))))
    :accessories (:tense-modal present
                  :progressive
                  :unmarked))
```

Figure 2.3: Bundle specification for *John is giving a book to Mary*

The input to MUMBLE consists of:

1. the basic constituents of the utterance (grouping of the conceptual input);
2. the functional relationships among the units in terms of syntactic head, predication and optional modifiers (abstract syntactic decisions); and
3. lexical heads are all hard-wired (lexical choice).

MUMBLE uses three levels of representation: *realisation specification* (input) which is mapped during realisation onto *surface structure* (recording all syntactic attachments between constituents) which in turn is mapped during phrase structure execution onto *word stream*. Figure 2.4 shows the phrase structure output of the realiser for the input in Figure 2.3.

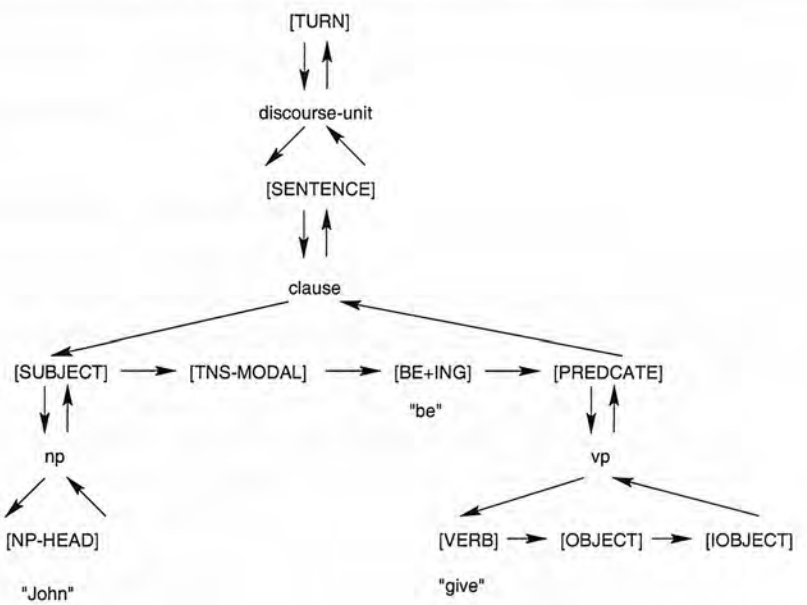


Figure 2.4: phrase structure tree generated by MUMBLE

The mapping from the input to the phrase structure is mainly realised by making choices in *realisation classes*. The realisation class is the main encoding of the decisions that the generator must make and therefore plays a similar role to systems in Systemic Grammar (SG) (as considered in implementations like NIGEL [Matthiessen 83]) and disjunctions in Functional Unification Grammar (FUG) [Kay 84] (as in the FUF system [Elhadad 93]). Because of the drive for psycholinguistic plausibility MUMBLE insists that all choices be deterministic and never undone. The grammar writer must

therefore pay special attention to the ordering of decisions, to avoid situations where an earlier decision leads to a dead-end. The mechanism to control the order of taking decisions is called the *bundle driver*. Bundle drivers are procedures that control how a set of constituents is attached to a head constituent. The control mechanism in MUMBLE is therefore completely procedural and tightly governed by the assumption of determinism. Because of the nature of its input MUMBLE does not have a high paraphrasing power.

2.4 Realisation as unification

This approach is based on Functional Unification Grammar [Kay 84, Kay 85] (previously known as 'Functional Grammar' [Kay 79]). The main characteristic of FUG is that all information is uniformly described using the same type of data structure — *functional description* (FD).

2.4.1 Functional descriptions

A functional description consists of *features* which have an *attribute* and an associated *value* (i.e., features are attribute-value pairs). Attributes are atoms; we will not be concerned with their internal structure. Values can be atomic. So CAT: s is a feature with attribute CAT (category) and value s (sentence). Values can also be non-atomic, namely other (embedded FDs), *constituents sets* and *patterns*. An FD is boolean expression over features. We distinguish conjuncts from disjuncts by the kind of brackets used to enclose their features: [ ] and { } respectively. It is a crucial property of FD that no attribute appears more than once in any conjunct (at the top level). If the value of an attribute is an FD the same applies to the embedded FD. Thus a sequence of attributes $\langle a_1, a_2, \ldots, a_n \rangle$ uniquely identifies a value. If the FD contains disjuncts then the value of identified by the path will naturally be also a disjunction. The values of two attributes can be made to be the same (we will make this more precise in a moment when we introduce unification). Notationally we write: ATTR_1: $\langle a, \ldots, \text{ATTR}_2 \rangle$ (i.e., the value of the first attribute is the path identifying the value of attribute ATTR_2).

Figure 2.5: Different kinds of functional descriptions

Figure 2.5 shows different kinds of functional descriptions where features are arranged vertically in so called *attribute-value matrix* (AVM) notation. The FD on the left shows attributes with non-atomic values. The FD represents the list ⟨ a, b, c ⟩. The FD in the middle represents disjunction. The FD on the right in Figure 2.5 contains embedded FDs and also the values of paths ⟨ SUBJ, AGR ⟩ is the same as the value of the path ⟨ HEAD, AGR ⟩. Visually in the AVM notation we have represented this as a *coreference* index $\boxed{1}$.

Constituent sets (values of the CSET attribute) are sets of paths identifying the constituents (but not their order!) of the of the top level FD. Thus, CSET specifies immediate dominance. Not surprisingly, it is the PATTERN attribute that constrains word order — the value of PATTERN is a regular expression over paths (see Figure 2.6) which in essence gives a partial ordering of the constituents which needs to be respected by the total order at the end. The PATTERN attribute specifies linear precedence.

2.4.2 Unification

The only mechanism allowed when dealing with FDs is *unification*. Intuitively, the unification of two FDs consists of building a larger FD that comprises both input FDs and is compatible with both: [ATTR1: a] unifies with [ATTR2: b] producing $\begin{bmatrix} \text{ATTR1:} & \text{a} \\ \text{ATTR2:} & \text{b} \end{bmatrix}$. Two FDs might be inconsistent in the sense that they might provide contradictory requirements for values of attributes and their unification might fail (e.g., [ATTR: c] and [ATTR: d]). Crucial features of the unification process are that it is:

1. independent of the order of features in the input FDs;
2. bidirectional (information flows both ways; both FDs can augment each other with new information);
3. monotonic (the order of subsequent unifications is irrelevant as to what the final result is);
4. declarative (the grammar is best viewed as a set of constraints to be added to or checked against an input).

Both the conceptual input and the grammar are represented as FDs. The input represented as a FD specifies the semantics and also provides abstract syntactic information (e.g., what category the input should be generated as). Here is a simple input FD:

$$\begin{bmatrix} \text{SEM:} & \begin{bmatrix} \text{ACTION:} & \text{run} \\ \text{ACTOR:} & \text{john} \end{bmatrix} \\ \text{CAT:} & \text{s} \end{bmatrix}$$

The grammar is also a FD (which contains disjunctions/alternations for clauses which describe major categories) and represents possible syntactic semantic combinations (see Figure 2.6).

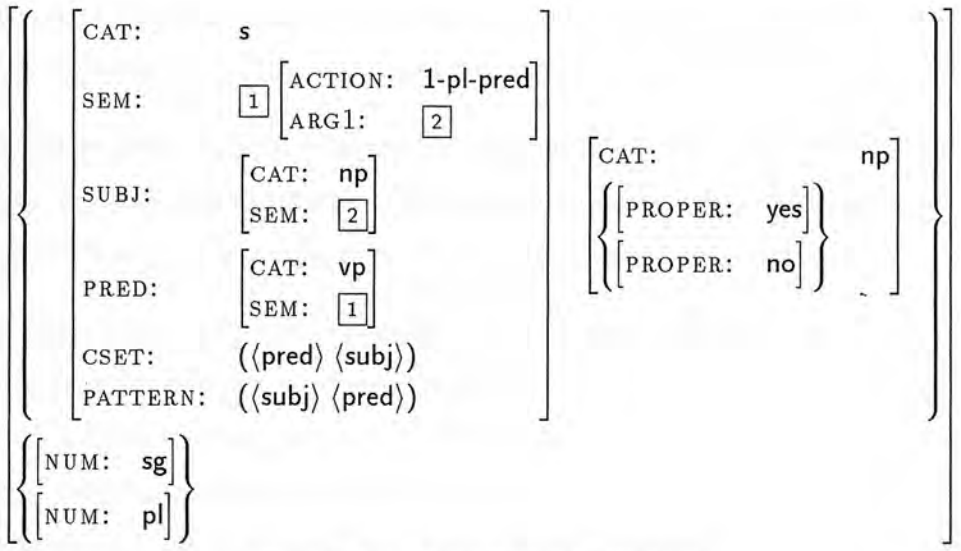


Figure 2.6: Example FUG grammar

The generation proceeds as follows:

1. Unify input with grammar:

   The unification algorithm begins by selecting the syntactic category from the input FD and unifies the grammar for that category with the input FD. Unification is controlled by the grammar and basically consists of checking grammar attribute-value pairs of this category against the input. If a grammar attribute does not exist in the input FD, the grammar attribute-value pair is added to the input FD. If the attribute does exist, the grammar and the input FDs' values for this attribute are unified and the results are added to the input. This stage of unification can be characterised as a breadth first sweep through the top level category adding restrictions governed by this category.

2. Identify values of syntactic constituents:

   The constituents that are to be elaborated are specified by the `cset` attribute. The order in which they appear in the `cset` specifies the processing order (see next step). The actual surface order is specified by the `pattern` attribute.

3. Unify each constituent with grammar:

   Each constituent of the resulting FD is in turn recursively unified with (a new copy of) the grammar in the same way. At this stage, unification results in successive refinement of embedded constituents.

Decision making is top down but not necessarily left-to-right. A further distinction is that all decisions get made at the top level before moving to embedded constituents.

As we can see from Figure 2.6 the generation is driven by what the grammar allows for which is reminiscent of the grammar-driven approach in Systemic Grammar. However, because unification is a very general operation this need not be the case.

The latest generation system based on FUG is Michael Elhadad's FUF. [Elhadad 93] presents techniques in lexical choice that allow for the generation of more fluent text than was previously possible. Lexical choice is viewed as the interface in a generation program between a conceptual representation expressing what information is to be conveyed by the text, and a linguistic representation, where all open class lexical items are selected and lexical dependencies are specified. The focus in the ADVISOR-II system which used the FUF realisation component is on evaluative expressions to satisfy a speaker's argumentative intent. Choosing an evaluative expression directly corresponds to a pragmatic goal. The same argumentative intent can be realised by a variety of lexical items at different syntactic ranks.

FUF introduces a number of extensions to the basic mechanisms in FUG:

1. support for types and inheritance;
2. type specific unification procedures;
3. support for modular grammars and interaction with external knowledge bases based on the ability to call procedures in other programming languages;
4. indexing of the grammar;
5. new control mechanisms allowing the implementation of larger and more complex grammars and the efficient processing of floating constraints.

Increasing the paraphrasing power is a main motivation for FUF. This implies that the input to the surface generation module must be general enough to account for many

different syntactic and lexical decisions. When paraphrasing is limited, in certain situations a generator can simply be unable to produce text.

The semantic representation, by virtue of being represented as a typed feature term, still contains the immediate dominance relations between elements. In order to tackle the head switching problem in a principled way, restructuring of feature terms must be used (inference, rewriting) [Dorna *et al.* 98].

FUF expects that lexical choice is given in the input and thus handling lexical gaps is problematic.

FUF is also non-deterministic and while it attempts to do clever backtracking (borrowing ideas from logic programming) it is prone to redoing previously successful computations.

The uniform representation of all information levels has proved to be convenient but it is not clear whether, for example, syntactic and semantic structures are subject to the same constraints and thus should be represented uniformly.

A number of very influential systems have used Functional Unification Grammar: TEXT [McKeown 85], TELEGRAM [Appelt 85], COMET [McKeown *et al.* 90], FUF [Elhadad 91]. Unification is central to other linguistic frameworks: Lexical Functional Grammar (LFG) [Bresnan & Kaplan 82] and Head-Phrase Structure Grammar (HPSG) [Pollard & Sag 87], [Pollard & Sag 94]. The class of such grammars is referred to as unification-based. Generation within LFG has been considered in [Kohl 91, Kohl & Momma 92, Kohl 92, Wedekind 88]. For a comprehensive study of the logical aspects of feature structures see [Carpenter 92].

2.5 Realisation as classification

Another very general operation (like unification) is that of classification which recently has been used for generation too [Mellish 91].

In the mid 80s a family of knowledge representation formalisms was developed which were based on structured inheritance hierarchies (taxonomies) to reason about knowledge. An important aspect of these systems was that they supported automatic classi-

fication to place a description of an object with respect to the hierarchy. This family is known as the KL-ONE family [Brachman & Schmolze 85] and representative formalisms include: CLASSIC and LOOM [MacGregor 90]. Classification systems have been used for representing the knowledge base of generators (because they are an expressive formalism).[2] Yet, classification can be used to perform the task of surface realisation as well.

Classification systems are based on a taxonomy of objects/classes/concepts and classify a new object in the taxonomy.

Taxonomies are semantic networks that organise knowledge according to levels of generality. The taxonomy consists of a set of classes/concepts connected by IS-A links. A subclass inherits information from its superclasses, and a super class is said to subsume its subclasses.

Concepts are descriptions with potentially complex structure. Complex concepts have *roles* which are used to relate concepts to other kind of knowledge (including other concepts). Role fillers describe value restrictions for concepts and are either concepts or constants. Values of roles can be inherited from the parent/super classes or overridden.

In classification-based systems a distinction is made between *definitional* and *assertional* roles. The definitional roles are used to describe the structure of objects and assertional roles are used to record additional information about concepts. This distinction is drawn in order to support efficient types of reasoning. The component used in formalising the structure of objects is referred to as the terminological component (*T-box*) and the component for adding extra facts about concepts as the assertional component (*A-box*). Normally the T-box is monotonic while the A-box non-monotonic. Non-monotonicity and the mechanism of inheritance in the assertional component is useful for describing sub-regularity/exceptions—if a value is not specified it is inherited from the parents; if it is specified it overrides the default value.

We give some examples of definitions using the notation of the PROLOG version of the I1 classification system which was implemented by Chris Mellish.[3]

---

[2] Classification systems can be used as a programming language [Mellish & Reiter 93].

[3] The LISP-based I1 system was the basis of a large Intelligent Document Advisory System (IDAS)

Primitive concepts are the initial building blocks of taxonomies and allow the specification of other concepts.

class1 $\ll$ *superclass.* % *class1* is a primitive class of *superclass.*

Defined concepts are specified by means of concept forming operations that link together other concepts. There are a number of ways in which this can be done: restricting one or more parents, tightening the conditions inherited from parents, pointing to more than one parent without the need for structural conditions (i.e., definitional roles).

class2 $\equiv$ {*parent1, parent2*} with [*role1: # atom1,*
role2: class37].
class2 $\Rightarrow$ [*role3: @*$\rightarrow$*role1,*
role4: class with [*role: class4*]].

Atomic values are prefixed by '*#*' as in *# atom1* so that the system can distinguish them from class names (like *class37*). Roles *role1* and *role2* are definitional; roles *role3* and *role4* — assertional. The $\equiv$ line/specification of *class2* is a definition and the $\Rightarrow$ line/specification is an assertion.

2.5.1 Classification

Classification is the process of taking a description of an object and finding its place in the taxonomy. This involves finding the most specific concepts that subsume the input and the most general concepts subsumed by the input.

In order for a description/class to be classified under a class (e.g., *class2* above) it must have all the definitional roles and the corresponding values (or more specific) of the parents (*parent1* and *parent2*). The input description also must have a role *role1* with a value *# atom1* and a role *role2* with a value more specific than *class37*.

The assertional definitions are ignored during the classification process but are used later on. Once the place of an object in the hierarchy is found all properties of this object are known (can be inferred). The properties that will concern us will be in the assertional component (i.e., the values of assertional roles) and these are inferred from the parents and other superclasses by a process of *default inheritance*:

Suppose we have the following definitions:

superclass $\Rightarrow$ [*role: # a*].
parent $\equiv$ {*superclass*} with [...].
class $\equiv$ {*parent*} with [...].

class will inherit the value of *role* from its *superclass* (providing none of the intervening *parents* change it). However if class *class* had an assertional statement like:

[Reiter *et al.* 92].

class ⇒ [*role:* # *b*].

it will override the value of *role* to # *b*. This overriding of default values that are inherited from ancesstor classes introduces non-monotonic behaviour and is not unlike the techniques of object-oriented languages. Assertional roles are evaluated in a lazy fashion.

2.5.2 Generation through classification

The input is represented as a class relating the semantics to the top-level syntactic category (or syntactic constraints in general):

input_sem ≡ {*focus_obj*} with [*pred: love*
agnt: john
obj: mary].
class ≡ {*sent*} with [*sem: input_sem*].

The grammar is represented in the taxonomy (see Figure 2.7).

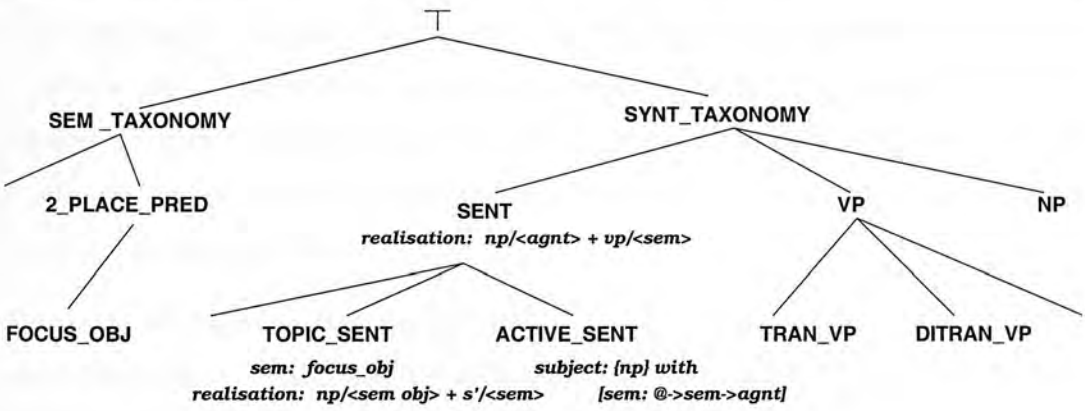


Figure 2.7: Generation with classification

The generation cycle for classification is [Mellish 91, p113]:

1. Classify current description (a syntactic object with certain semantics).
2. (By inheritance) find those features whose values are needed to get the realisation of this phrase.
3. (By inheritance) find the descriptions of the values of these features.
4. Iterate on each of these descriptions.

Note the similarities with the generation cycle for unification grammars. In the above example, the input *class* with semantics *input_sem* is classified under TOPIC_SENT

as TOPIC_SENT has a definitional role *sem* which states that TOPIC_SENT subsumes classes with *sem* role more set to *focus_obj* (or a more specific class) which is the case for the input class. The final value of the realisation (assertional) role (of the classified input) is: *Mary, John loves.* Advantages of generation by classification include:

- Single representation and reasoning component for both domain and linguistic knowledge (difficult for unification and systemics).
- Efficient *indexing* mechanism (on semantic grounds).
- Default inheritance allows for stating generalisations (and in a compact form). Properties of a class (semantic structures or linguistic objects) to be expressed at the most abstract level possible.

Current classification-based generators are deterministic. They select a single word corresponding to the most specific concept. The top-down algorithm fails in case of lexical gaps. Determinism also implies single syntactic constructions. But it is possible to have the generator produce an encoding from which the different word orders can be recovered (cf. handling free word order phenomena in German within CL-ONE, [Manandhar 94]).

The semantics corresponding to the generated sentence cannot be more specific than the semantic input. Sometimes a class is classified too high up in the hierarchy. Additions to the input semantics forced by the language are ruled out .

Generation with classification has been pursued from a multilingual perspective: French [Bellos 92], German [Klein 96], Turkish [Dick 93], Romanian morphology [Cristea 93], lexical choice in Arabic [Al-Jabri 97], sentence generation in Russian [Gromova *et al.* 96].

2.6 Semantic head-driven generation (SHDG)

The semantic head-driven algorithm (SHDG) [Shieber *et al.* 90, vanNoord 90] generates strings from logical form encodings. It significantly improves upon previous bottom-up methods based on Earley deduction [Shieber 88] in that it places fewer restrictions on the class of grammars to which it is applicable. In particular, it allows use of

semantically non-monotonic grammars, yet unlike top-down methods, it also permits left recursion. The enabling design feature of the algorithm is its implicit traversal of the analysis tree for the string being generated in a semantic, head-driven fashion. The algorithm reflects a traversal strategy respecting the semantic structure of the string being generated, rather than the string itself, i.e., the order of processing is geared towards the logical form of the input (head-driven) and the information available in the lexical entries (bottom-up).[4]

2.6.1 How it works

The input is a pair containing the syntactic category that we want to generate (initially a sentence) and the logical form that we want to verbalise: $\langle s, X \rangle$ (see Figure 2.8 which is from [König 94]).

In the top-down (prediction) step (labeled as *lex* in Figure 2.8) a lexical entry (or more generally a pivot) is chosen whose logical form (LF) matches the LF of the input (i.e., the two syntax-semantics pairs are linkable and the actual definition of the *link* relation is $link(\langle x, X \rangle, \langle x_i, X \rangle)$ and x_i is a head-corner of x). The pivot is reached starting from the original $\langle s, X \rangle$ pair, matching it against the LHS of a (CFG) rule, then hopping to a head daughter of the rule which shares the same semantics as the mother (such rules are called chain rules). This head daughter is made our new goal and we recurse until we reach a rule (or a lexical definition) which does not have a daughter sharing the semantics with the mother (non-chain rule). The mother node of this last rule/lex. definition is called the pivot.

Then (in the expansion step[5]), we try to build a phrase such that the word which was found in the previous step is a head word for this phrase. Looking for ways to expand the current structure the algorithm has to find a grammar rule which has the current structure as a head constituent in the RHS and try to generate the sister daughters of the head in this rule—the mother node of this rule will be the new

[4] Although the description of the in Semantic-Head-Driven Generation in [Shieber *et al.* 90] and [vanNoord 90] is cast in Prolog terms and illustrated with Prolog code, the generation strategy is more general and we present a more 'abstract' description of it.

[5] In the description of the semantic head-driven generation algorithm this step is called connecting the current structure (initially lexical head) to the *s*-node in the syntactic tree being built.

all leaves are labeled with terminals and the tree does not contain any dotted lines *(global-success)*

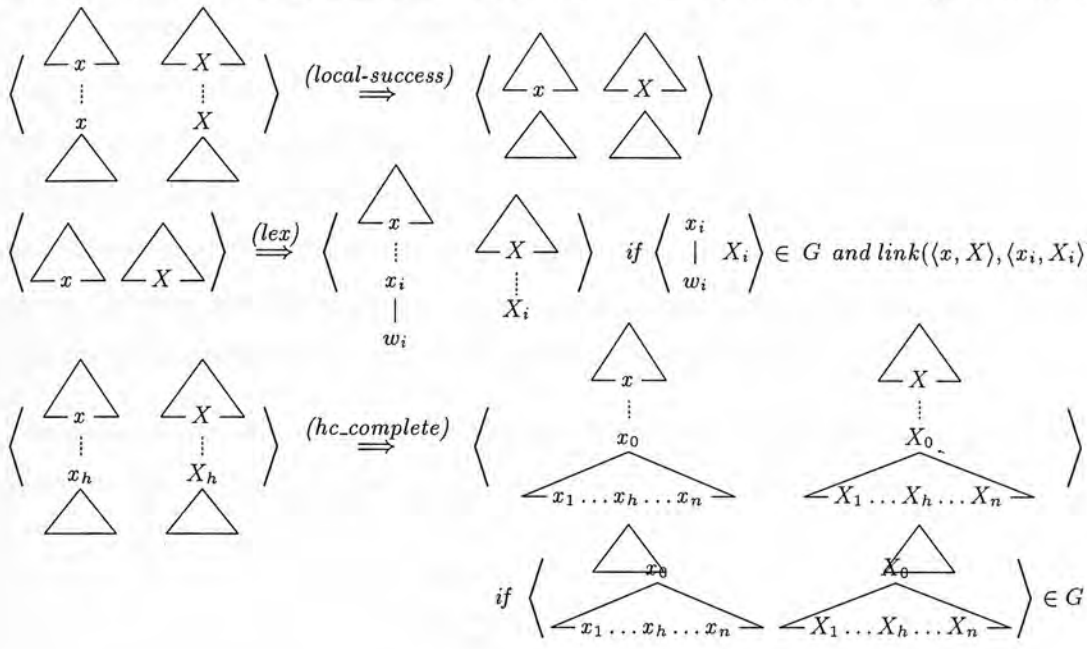


Figure 2.8: Head-corner generator
(G grammar description; x_i syntactic category; X_i semantic representation)

expanded structure. In Figure 2.8 this is the rule *(hc-complete)* which performs a 'head-corner' completion step for a (linked) phrase x_h, which leads to the prediction of the head's sisters. Connecting the current node to the original goal is performed in a bottom-up fashion. First, a bigger syntactic structure is built from the initially found word, then recursively this bigger structure is expanded (using the same expansion step) finally giving the sentence structure.

The algorithm finishes when the current syntax-semantics pair is identical to the original goal (rule *local-success* in Figure 2.8).

Generation succeeds if all the leaves of the syntax tree are labeled with terminals and if no link markings exist (rule *global-success* in Figure 2.8).

2.6.2 Example

As an illustration we present a simplified example: *John kisses Mary.*

The relation between meaning and linguistic form is stated using rules like:

```
node(s/LF, P0-PN) --->                    % S -> NP VP rule
  [ node(vp(Subj,Comps)/LF  , P1-PN),     % head listed first; Comps in LF
    node(np              /Subj, P0-P1) ]. % subj sem determined from VP
```

This is the equivalent of the S → NP VP rule. The syntactic categories are coupled with the semantic form corresponding to them (giving the *Syn/Sem* pair). The immediate dominance relation is made explicit between the left-handside *node* of the `--->` arrow and the *node*s to the right of it (represented as a list). The head *node* in these rules is always listed as the first daughter which motivates the augmenting of the *node*s with a second argument to mark constituent ordering (using difference lists).

The nodes in the derivation tree in Figure 2.9 are either terminal nodes (like *john*, *kisses* and *mary*) or *Syn/Sem* pairs.

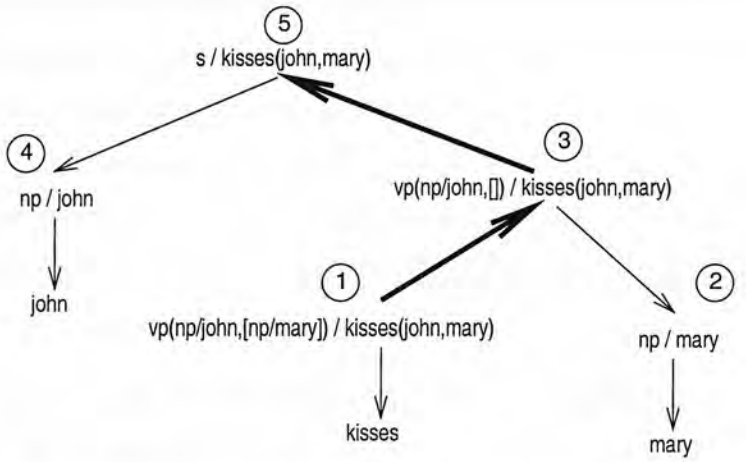


Figure 2.9: Example of semantic head-driven generation

Predicting the lexical head for the semantics *kisses*(*john*, *mary*) suggests the verb *kisses*.

```
node(v(X,[Y])/kiss(X,Y), [kisses|L]-L).
```
[6]

Then using the *vp*-complementation rule the verb is expanded to incorporate its object yielding a bigger *vp*-structure. Here we have used the fact that *kisses* is a head for the verb phrase *kisses mary*. Finally using the S → NP VP rule the VP is expanded to a sentence. The numbers in the circles in Figure 2.9 show the actual traversal of the derivation tree.

[6] Note how this lexical definition dismatles the semantics and determines the semantics of the subject (`X`) and the semantics of the complements (in this transitive construction only one) `[Y]`.

2.6.3 Discussion

Generation is head-driven (the initial word we start building our sentences from is a semantic head; in expanding smaller structures into bigger we use rules such that the smaller structure is a semantic head for the mother node) and also bottom-up as we start to build the sentence gradually expanding the initially predicted word into a full-blown sentence. If the set of chain rules is empty, SHDG operates strictly top-down. If the set of *non*-chain rules is empty, SHDG operates purely bottom-up.

SHDG as formulated makes implicit assumptions about the form of the grammar. The notion of semantic head relies on rules with one RHS daughter having the same semantics as the LHS. But does a VP really mean the same as its sentence? Another way of thinking of rules is as structure building instructions/statements where the rule contains information of how to put together the semantics of the daughter nodes to come up with the semantics of the mother. In this case it is not obvious why there should be a daughter which has *identical* semantics as the mother. That is not to say that there shouldn't be a main daughter but is rather a comment on the balance between the information about the structure building/decomposition associated with the rule and the semantics of the main daughter.

SHDG is a heuristic approach. It is correct, but not necessarily complete (in a PROLOG implementation) or efficient.

Semantic head-driven generation says very little about how lexical choice is to be performed. In particular, there is a potential non-determinism in the choice of the initially predicted lexical head. A wrong choice at this point will be rejected later on during the execution of the algorithm and alternatives will be found upon backtracking (considering other choices) [Samuelsson 97].

A more general comment on the algorithm is that it is in fact a grammar-driven approach to the relation *meaning-form* (because of the particular ways the rules are being encoded—phrase structure rules augmented with semantic information about each constituent).

Due to the assumption about the hierarchical nature of the input (namely PROLOG

terms) SHDG cannot produce different ordering of modifiers, or incorporate modifiers in order other than in the order in which they 'wrap' the semantics (recall the discussion on page 11), nor produce modifiers after the head noun is chosen. Modification rules are non-chain and the modifiers are produced before the head noun in a top-down fashion. This does not allow for proper treatment of co-locations.

We have defined an alternative strategy for generation with constraint-based grammars which assumes a list of modifiers in the semantic representation and our strategy addresses all the points above and can be used with HPSG-like grammars [Nicolov & Mellish 96]. We also allow for a language dependent ordering of modifiers. Our alternative semantic head-driven algorithm can be seen as a scaled down version of the generation system discussed in this thesis. A number of NLP systems use SHDG in their generation components, e.g., CORE LANGUAGE ENGINE [Alshawi 92], MIMO2 [vanNoord *et al.* 91].

2.7 Generation from non-hierarchical representations

As we saw in the previous section on semantic head-driven generation the system cannot produce certain paraphrases because of the hierarchical nature of the input. This of course doesn't have to be the case and there are systems that do not assume hierarchical input and use network representations. The system described in this thesis is such one. In our work we use Conceptual Graphs [Sowa 84, Sowa 92] as a semantic network foralism to encode the input to our generator. Other formalisms have also been considered in particular in work on natural language semantics, namely Underspecified Discourse Representation Theory (UDRT) [Reyle 93] and Minimal Recursion Semantics (MRS) [Copestake *et al.* 97] and the advantages of Conceptual Graphs for generation are shared by UDRT and MRS.

The use of semantic networks in generation is not new [Simmons & Slocum 72], [Shapiro 82]. Two main approaches have been employed for generation from semantic networks: *utterance path traversal* and *incremental consumption.*[7]

[7] Here the incremental consumption approach does not refer to incremental generation!

Utterance path approach. An utterance path is the sequence of nodes and arcs that are traversed in the process of mapping a graph to a sentence. Generation is performed by finding a cyclic path in the graph which visits each node at least once. If a node is visited more than once, grammar rules determine when and how much of its content will be uttered [Sowa 84]. It is not surprising that the early approaches to generation from semantic networks employed the notion of an utterance path—the then popular grammatical framework (Augmented Transition Networks) also involved a notion of traversal path.

The utterance path approach imposes unnecessary restrictions on the resources (i.e., that the generator can look at a limited portion of the input—usually the concepts of a single relation); this imposes a local view of the generation process. In addition a directionality of processing is introduced which is difficult to motivate; sometimes linguistic knowledge is used to traverse the network (adverbs of manner are to be visited before adverbs of time); finally stating the relation between syntax and semantics involves the notion of how many times a concept has been visited.

Incremental consumption. Under the second approach, that of incremental consumption, generation is done by gradually relating (consuming) pieces of the input semantics to linguistic structure [Boyer & Lapalme 85, Nogier 91]. Such covering of the semantic structure avoids some of the limitations of the utterance path approach and is also the general mechanism we have adopted (we do not rely on the directionality of the conceptual relations *per se*—the primitive operation that we use when consuming pieces of the input semantics is maximal join which is akin to pattern matching). The borderline between the two paradigms is not clear-cut. Some researchers [Smith *et al.* 94] are looking at finding an appropriate sequence of expansions of concepts and reductions of subparts of the semantic network until all concepts have realisations in the language. Others assume all concepts are expressible and try to substitute syntactic relations for conceptual relations [Antonacci & Smith 92].

Other work addressing surface realisation from semantic networks includes: generation using Meaning-Text Theory (MTT) [Iordanskaja & *et al.* 91], generation using the SNePS representation formalism [Shapiro & Rapaport 90], generation from concep-

tual dependency graphs [vanRijn 91], generation using Lexical Conceptual Grammar [Oh *et al.* 92], generating from CGs using categorial grammar in the domain of technical documentation [Svenberg 94], translation tools with a generation component — DB-MAT [Bontcheva 96]. Yet, the generation techniques that have been used do not constitute novel approaches and we do not review them here.

2.8 Generation as incremental consumption

This approach focuses on the resource-based notion of generation where the generator explores the input semantics and relates it to syntactic structure remembering that it has expressed (consumed, covered) this bit of semantics and subsequently this portion of the input need not be realised (but might have to be kept for connectivity reasons, i.e., connecting subparts of the conceptual input that will otherwise remain separate).

[Nogier 90] and [Nogier & Zock 92] describe lexical choice as a process of pattern matching. An important assumption that is made is that words, sentences and texts are:

- just different units to convey a message (words are simply shorthand labels for larger conceptual structures);
- represented in the same way (i.e., using the same formalism)

The focus of this research is on the process: lexical choice is modelled by matching word definitions with the initial message (utterance specification). Similarly syntactic decisions are made by matching larger conceptual configurations against syntactic patterns. Conceptual graphs are used as a representation formalism on all levels.

The authors give a lexicalisation example which covers the whole process of sentence generation. We briefly reproduce it in order to give an idea of the generation algorithm.

The input is a message specification encoded as an utterance graph (cf. Figure 2.10).

The algorithm consists of the following steps:

1. Key-word retrieval: a lexical pool is selected from the lexicon such that all words contain a certain *key* concept in their definitions

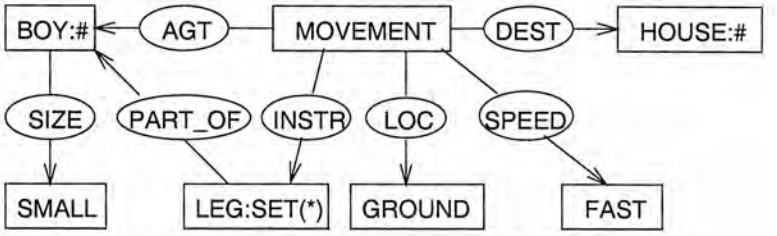


Figure 2.10: An utterance graph (message)

2. Pattern matching with the lexical pool: among the words from the lexical pool only those that are consistent with the information in the initial utterance graph are considered

3. Choosing the best candidate using a corelation factor: this is a measure which gives the number of concepts of the word definition (note that all these concepts have to appear in the utterance graph for the word to be consistent with it)

4. Instantiating a syntactic structure: the skeleton syntactic pattern suggested by the word is instantiated with the additional information from the utterance graph

5. Replacing the subgraph covered by the syntactic structure in the initial utterance graph with the (instantiated) syntactic graph.

Given a semantics like the one in Figure 2.10 key-word retrieval forms a lexical pool of words containing the concept [MOVEMENT] in their definition (movement verbs): *walk*, *drive*, *move*, *swim*, *run*, *fly*, etc. These are quite a few and probably not all of them are consistent with the initial utterance graph. This is handled by the pattern matching in the second step. There still might be a couple of words that can pass through this filter (like *move*, *walk*, *run*). A conciseness measure is used to determine the candidate that incorporates most concepts from the initial semantics—*run*. A generic syntactic structure for an intransitive verb like *run* is:

```
<NOUN:AGENT> <<-=( SUBJ )== <VERB:ACTION>
```

Instantiating this structure would yield: `["boy"]<-(SUBJ)-["to run"]`. which when substituted for the subgraph in the initial utterance corresponding to its semantic content simplifies it considerably producing a structure closer to the surface form.

1. The approach avoids redundancies (*He walked on the ground.*) and contradictions (*He swam in the sky.*)

2. The approach gives a potential for treating the *carry over* phenomenon—in spontaneous discourse people often cannot express all the information they had planned. Sometimes this information has to be encoded in the next utterance.

3. The approach supports paraphrasing in a natural way though the issue of which alternative is best suited in a given context still needs to be investigated.

4. Graph matching can be used to explain the differences between words.

Another situation, quite likely to occur in generation, is when the definition graphs for the words in the lexical pool add information to the initial utterance. In such cases the *carry over* might include goals for undoing strong commitments made in the current sentence. Also the measure for choosing a best candidate from the lexical pool will have to be complicated. Even worse constructing the lexical pool in the first place might not be as trivial then. Similar problems are also discussed in [Barnett *et al.* 94].

A good feature of this approach is the blurring of the difference between words and phrases (sentences) and viewing them simply as different sized units to convey the meaning. This is important for multilinguality as different languages have different potential for packaging semantic information in words, clauses, and sentences.

From an implementation point of view it is not necessary for every word to contain as many conceptual graphs as there are corresponding syntactic structures in which the word could participate. The semantics could be represented once and the individual concepts can be referred to from the particular syntactic structures. This allows for faster retrieval as the semantic matching needs to be performed only once.

2.9 Procedural approaches: Semantic Syntax

In this section we briefly review a non-standard, yet highly sophisticated approach to sentence generation. While in previous approaches the inputs have not been really logical representations as used in format semantics this approach makes no compromises

in this respect. By design the framework is procedural. Much more procedural than the PENMAN implementation of Systemic Grammar. Yet, the approach does address head switching and substantial grammars have been developed for different languages.

Semantic Syntax [Seuren 93] is a theory of syntax which tries to establish a (bidirectional) mapping between the meaning representation of sentences and their surface realisation. "Semantic Syntax makes for a subtler and more precise coverage of the facts of the languages treated than any other grammar system." [Seuren 93]

In the following we will look at the generation component and the machinery that SESYN (a system that implements the Semantic Syntax framework) uses in order to tackle translation mismatches.

During generation a language specific semantic structure (called semantic analysis structure) is being transformed into a sentence (surface structure). The semantic analysis structures are higher-order predicate calculus trees. The surface structures are based on an orthodox version of Transformational grammar (as it was known up till the 1970s).

Generation starts by constructing a semantic analysis structure (SA). This is done on the basis of a set of formation rules which specify what the valid semantic analysis structures are. The formation rules are a grammar for the semantic analysis structures/trees.

Then, the SA structure is transformed into a surface structure using transformational rules. Transformational rules are of two types: rules that apply cyclically and those that' don't. The cyclic rules apply in a bottom-up way starting with the most deeply embedded S and ending with the top-S. They are mostly lexicon-driven: predicates are lexically marked for the cyclic rules they induce. The post-cyclic rules are largely structure-driven and apply in linear order as defined by the grammar. There is a limited number of highly constrained transformation rules.

In order to handle structurally different translation equivalents between two languages SESYN postulates the existence of a higher semantic level—language independent semantic analysis (LISA). This level of representation mediates the translation between the SA structure of the source language into the SA structure of the target language.

In addition a semantic lexicon is used which bidirectionally maps partial SA structures or just elements from the SA structure onto language independent semantic analysis structures.

As an example we illustrate the generation of the sentence: *"He likes to swim."* from a language independent semantic analysis structure which mediates the translation of the target English sentence above from its Dutch equivalent: *"Hij zwemt graag."* This pair of translation equivalents exhibits a translation mismatch of head switching. Seuren in [Seuren 93] shows how the translation from Dutch into English proceeds assuming a parsing component that delivers the Dutch SA. A sketch of the semantic lexicon and the processes to relate SA structures to language independent semantic analysis structures is also given.

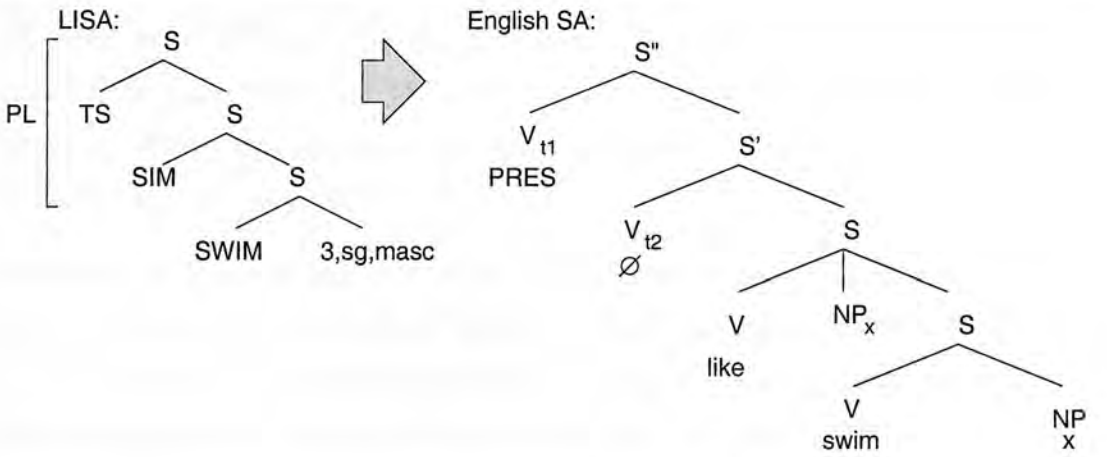


Figure 2.11: Relating a language-independent to a language-specific semantic analysis

The semantic lexicon translates PL (with pleasure) into English as the lexical verb *like*. The entry in the semantic lexicon says that PL is scope-insensitive with respect to the tenses (positions TS and SIM), as indicated in the LISA-tree. This gives it the freedom to occur below the two tenses in the English SA. The tenses translate trivially:

TS (time of speaking) $\Leftrightarrow$ PRES

SIM (same time as) $\Leftrightarrow V_{t_2}$

The S-structure $_S$[SWIM 3,sg,masc] is treated as follows. As specified in the English

lexicon, the verb *like* takes an NP-subject and induces the cyclic rule of subject deletion. The S-structure $_S$[SWIM 3,sg,masc] has to be attached to *like* as object complement-S. The procedure is such that the subject of SWIM, i.e. (3,sg,masc), is transferred to *like* and will then control the application of subject deletion on the *like*-cycle. The resulting English SA (cf. Fig 2.11) is the input to the English transformation component. The transformations flatten the tree and move certain constituents.

Comments

SA-trees are not entirely universal. They are to some extent still language-specific in so far as different structures express identical meanings (eg. He could have eaten vs Dutch hij had kunnen eten). It is claimed that the structural differences must be manifest at SA-level if the syntax is to remain non-arbitrary. This difference between SA structures is also due to the fact that SA structures already have the lexical items specified in them. Thus, the paraphrasing power of SeSyn is limited to producing different surface structures using the same words. The problem of lexical choice is currently not addressed within SeSyn.

SeSyn is a procedural framework: the transformational component plays a central role. In SeSyn there is no surface syntax grammar. Instead a grammar for the SA structure is used coupled with a transformational component. The syntactic information in the lexicon specifies only the syntactic arguments required by the particular lexical item. Thus, the surface syntax is specified indirectly by the grammar for the semantic analysis structures and the transformations.

Language-specific differences amount largely to different post-cyclic rule orderings, different parameter settings in otherwise identical rules, different lexical inductions of cyclic rules, or different positions in the formation rules. It is possible to envisage a parametrisation of the general theory when more grammars of different languages are available (currently English, French, German and Dutch grammars are available).

2.10 Conclusions

In this chapter we surveyed a wide range of state-of-the-art approaches to NLG. Most of the approaches assume a tree-like semantics and a tight relation between syntax and semantics. The non-deterministic approaches do not avoid duplication of computations. In generating alternative solutions, they cannot give preference to better ones. Generation of the most important part of the semantics first for time-out scenarios was not considered at all. Most approaches expect lexical decisions to be made in advance.

The incremental consumption approach was our intellectual departure. It uses a non-hierarchical conceptual representation which we consider important in order for a generator to have a high paraphrasing power. Instead of keeping a uniform representation and having mixed intermediate representations assuming non-monotonic mechanisms we provide a mapping from the input conceptual representation (Conceptual Graphs) to syntactic structures in a linguistic theory (D-Tree Grammars). It is exactly the formalisms of Conceptual Graphs and D-Tree Grammars that we consider in the next two chapters.

Chapter 3

Non-Hierarchical Representation Systems

In this chapter we present the formalism of conceptual graphs which we use to encode the input to the generator. Conceptual graphs are a non-hierarchical representation formalism and they have well defined deductive mechanisms. They have been developed by John Sowa [Sowa 84] but stem from the tradition of the existential graphs of Charles Sanders Pierce [Hartshorne *et al.* 58, Pierce 93]. Our goal here is not to go through all aspects of the formalism of conceptual graphs but rather to give a flavour of it. Our contribution regarding the formalism of conceptual graphs has been in developing a representation for conceptual graphs which allows for destructive maximal join of two graphs such that both become further instantiated to the result of the maximal projection. Our idea stems from PROLOG implementation of unification of feature structures though the maximal join of two graphs is a considerably more complex operation, in terms of implementation.

Of course inputs to generators have different formats and here is a list of some of the widely used representational frameworks for representing the input to natural language generators:

- First Order Predicate Logic (FOPL) [Saint-Dizier 89];
- Sentence Planning Language (SPL) [Reiter *et al.* 92];
- D-structures (GB framework)

- Discourse Representation Theory (DRT) [Kamp & Reyle 93], λ-DRT
- Situation semantics [Barwise & Perry 83, Barwise *et al.* 91]; and
- Minimal Recursion Semantics (MRS) which is used in the Verbmobil project and by the CSLI group [Copestake *et al.* 97].

Some of the above frameworks are not logical notations.

3.1 Conceptual Graphs

Although conceptual graphs (CGs) have been developed as a system to represent natural language semantics there has been little work on language generation from CGs.

To meet the objections to standard logic, conceptual graphs have been designed as a more natural notation for logic. A lot of frameworks have tried to compensate for some weak features of symbolic logic. Common arguments against standard First Order Predicate Logic are:

- Not well suited to represent and handle semantics
- Symbolic Logic is existential
- Deductive reasoning is not common sense reasoning
- Improper use of variables and conjunctions

John Sowa has argued that "Logical Form should be tailored to linguistic form in order to avoid unnecessary complications in the grammar." [Sowa 84].

CGs are a declarative representation (as opposed to procedural knowledge representation which is difficult to maintain) and in their graphical form are a good compromise between semantic and cognitive representation.

> "Knowledge representations should be directed towards models that are sufficiently accurate in handling domain complexity yet remain understandable by human domain experts." [Polovina 92].

Last but not least, CGs are one of the most formalised among semantic network approaches – they are a complete notation for first-order logic with direct extensions to modal and higher-order logics.

Introductory texts on conceptual graphs include: [Sowa 84, Sowa 92, Way 91].

3.1.1 Conceptual Graphs

Definition: *Conceptual Graph*

A conceptual graph is a directed, finite, connected, bipartite graph [Sowa 84, p.73].

1. The two kinds of nodes of the bipartite graph are *concepts* and *conceptual relations*.
2. Every conceptual relation has one or more *arcs* each of which must be linked to some concept (a relation with n arcs is called $n-adic$). Every relation has just one outgoing arc. The number of ingoing arcs for an $n-adic$ relation is $n-1$.
3. A single concept is a conceptual graph. Every arc of every conceptual relation must be linked to some concept.

An example of a conceptual graph (in the graphical notation) is presented in Figure 3.1.

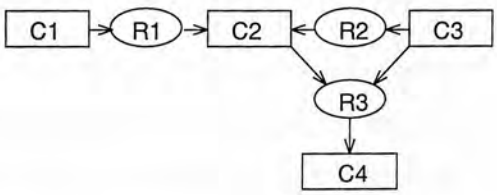


Figure 3.1: A conceptual graph

3.1.1.1 Conceptual Graphs as Graphs

In this section we define conceptual graphs in terms of the terminology adopted in graph theory and elaborate on the definitions.

A conceptual graph, like any graph, consists of a collection of nodes linked in certain ways. It is conventional to view a graph G as a pair of the sets of nodes N and another set describing the different ways the nodes are connected called the set of edges (or arcs) E. The notation is $G = \langle N, E \rangle$.

As conceptual graphs are bipartite graphs the set of nodes N consists of the union of two disjoint sets—the set of concepts C and the set of relations R ($N = C \cup R$).

Arcs will only be allowed to link nodes from the two sets, and not nodes within one set. Thus, arcs can be represented as as pair of nodes (x, y) where x and y belong to the disjoint sets C and R. And because conceptual graphs are directed, i.e., the arcs have arrows at one end, the order in which we represent the two nodes linked by an arc is important. That is to say, the pair (x, y) is ordered. Then the set of edges E is defined as:

$$E \subseteq \langle C \times R \rangle \cup \langle R \times C \rangle$$

The additional constraint that every relation node has just one outgoing arc and $n-1$ incoming arcs if it is an $n-adic$ relation is expressed as:

$$\forall r \in R \; (\exists! c \in C : (r, c) \in E)$$

The connectedness of the conceptual graphs means that starting from any node an arbitrary other node can be reached by following arcs (which are viewed as undirected):

$$\forall n_0, n_{k+1} \in N \; (\exists \{n_1\}_{i=1}^{k} \in N : (n_i, n_{i+1}) \in E \; or \; (n_{i+1}, n_i) \in E),$$

i.e., for the given nodes n_0 and n_{k+1} there exists a sequence of nodes $\{n_i\}_{i=1}^{k}$ such that adding node n_0 at the beginning and node n_{k+1} at the end of this sequence there is a longer sequence of nodes with the property that from every node the next one can be reached by following an arc or the reverse of an arc.[1]

In representing meaning with conceptual graphs, the concepts will stand for abstractions (like entities, actions, etc.) and the concepual relations will show how these abstractions are interrelated.

[1] If the ordered pair (x, y) is an arc then its reverse arc is (y, x).

3.1.2 Concepts

3.1.2.1 Type hierarchy

Just as in everyday language different concepts are thought as being of different types. Certain concepts are more general than others so it is natural to arrange all the concepts in a generalisation hierarchy. Water is a drink and apples are fruits, but both can be consumed. As it may happen that a certain concept is a specialisation of two different concepts the hierarchy should allow for concepts to have multiple dominating nodes (multiple inheritance): centaurs derive features from men and horses.

There is one type $\top$ which is a supertype of all types—i.e., everything is a subtype of $\top$. There is also an absurd type $\bot$ which does not represent any individuals.

Definition: *Type Hierarchy*

The type hierarchy is a partial order over the type labels. A type t_1 is a subtype of type t_2 is written: $t_1 \leq t_2$. The subtype relation is reflexive: every type t is a subtype of itself: $t \leq t$.

t is a *common subtype* of t_1 and t_2 if t is a subtype of both t_1 and t_2 ($t \leq t_1$ and $t \leq t_2$).

A *maximal common subtype* of t_1 and t_2 is a type $t_1 \cap t_2$ which is a common subtype of t_1 and t_2 s.t. for any subtype t of t_1 and t_2 $t \leq t_1 \cap t_2$.

Common supertype and *maximal common supertype* are defined analogously.

Figure 3.2 is a fragment of a description of a type hierarchy. Although we have shown a tree concept hierarchies have the structure of lattices.

3.1.2.2 Types and instances

An issue which has often been confused by novices is the distinction between a subtype and an instance (probably because they are often drawn in a similar way). An instance of a type is an individual of this type, while a subtype is just another name for a class of individuals having certain properties. It is not necessary for a type to have

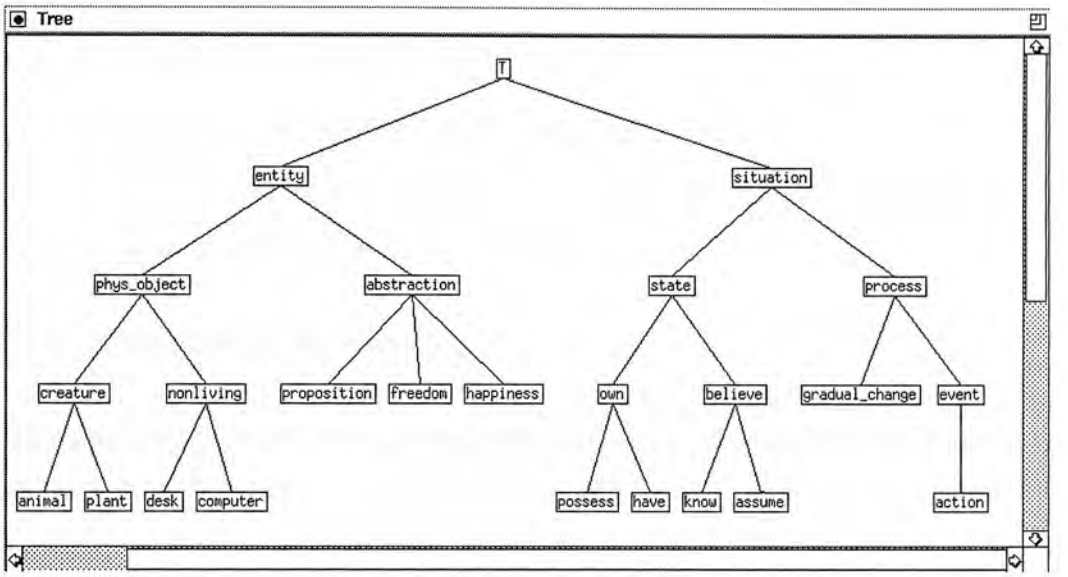


Figure 3.2: Graphical representation of a type hierarchy

representatives in the model that we are describing. Individuals correspond to logical constants and we will represent them as individual markers (the symbol # followed by an integer—*#40572*). All instances of all types are grouped in a set $I = \{$ *#40572, #691214, #800510,...*$\}$. Instances of a certain type are referred to as the denotation of this type—for example, the set of all computers is written as δCOMPUTER. As mentioned earlier, denotations of some types can be the empty set.

Concepts consist of a type label and an optional referent field. Here are some examples of concepts:

Example	Type	Referent
[CAT]	CAT	—
[CAT: #123]	CAT	individual marker #123
[CAT: *]	CAT	generic marker *
[CAT: *x]	CAT	variable *x

Definition: *Specialisation of concepts*

A concept c_1 is more specific than a concept c_2 if:

1. the type of c_1 is more specific than the type of c_2.
2. the referent of c_1 is an individual marker while the referent of c_2 is the generic marker * or c_2 has no referent.

3.1.3 Conceptual Relations

Conceptual relations link concepts and show that some relationship holds between the referents of the concepts. An n-adic relation will have n −1 incoming arcs and one outgoing arc. There are three kinds of conceptual relations:

1. *Primitive:* The primitive relation `LINK`. All other relations may be defined in terms of it.
2. *Starter set:* A starter set of conceptual relations will be used for representing natural language semantics. Nothing in the conceptual graph theory turns on the particular relations being used and the starter set can be changed as linguistic theory develops.
3. *Defined:* A relation can be defined as graph with particular concepts identified in it corresponding to the concepts that the relation can be connected. This mechanism allows a relation to be replaced by a graph and conversely a graph to be made more compact by replacing a subpart of it by a relation.

Although most of the relations in the starter set are dyadic (having two arcs) other relations with higher arity (i.e., with more arcs) can be used. In this case there is a need to distinguish the different incoming arcs and they are numbered.

Here are some examples of relations:

- Case relations (thematic roles): (AGNT), (PTNT), (STAT), (EXPR), (RCPT), (INST), (DEST), (RSLT);
- Spatial relations: (LOC), (IN), (ON), (ABOV);

- Attributes: (ATTR), (CHRC), (PART);
- Intersentential relations: (BFOR), (AFTR), (CAUS), (PURP), (METH), (AND), (OR);
- Mathematical relations: (MEAS), (>), (<), (=), (!=), (AVG), (CNT), (ARG1), (ARG2);
- Meta relations: (KIND), (SUBT), (DSCR), (STMT), (REPR).

3.1.4 Contexts and identity lines

Conceptual graphs can be embedded in one another.[2] Graphs need to be embedded in order to represent logical formulae which require that certain graphs be negated. Such embedding is useful in representing the semantics of natural language utterances too (cf. Figure 3.3).

Definition: *Context*

A context is a concept C whose referent field contains a non-blank conceptual graph g.

The graph g is said to be *immediately nested* in C, and any concept c of g is also said to be immediately nested in C.

Definition: *Nesting*

A concept c is said to be *nested* in a context C if either c is immediately nested in C or c is immediately nested in some context D that is nested in C.

- An *outermost context* is a context that is not nested in any other context.
- Two concepts c and d are said to be *conested* if either $c = d$ or there is some context C in which c and d are immediately nested.
- If a concept c is conested with a context C, then any concept d nested in C is said to be *more deeply nested* than c.

[2] Note the difference between the recursive embedding of typed feature structures and the embedding of conceptual graphs.

- A concept c is said to be *within the scope* of a concept d if either c is conested with d or c is more deeply nested than d.

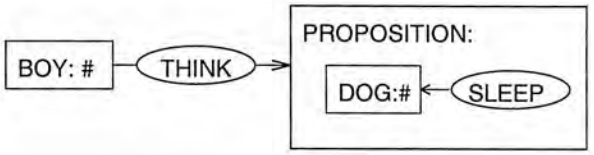


Figure 3.3: *The boy thinks the dog sleeps.*

Informally lines of identity are used to relate concepts (possibly in different contexts) and indicate that they refer to the same thing.

Definition: *Line of Identity*

A line of identity l is a connected graph whose nodes are concepts and whose arcs are called *coreference links.*

- If c and d are concepts in l where d is within the scope of c, then there must be a path of concepts $\langle c_1, \ldots, c_n \rangle$ in l, where the starting point $c_1 = c$, the ending point $c_n = d$, and each step $\langle c_i, c_{i+1} \rangle$ is a coreference link where c_{i+1} is in the scope of c_i.
- let c be any concept in l where every other concept d in l is within the scope of c. Then c is said to be a *candidate defining node* of l.
- If c is any candidate defining node of l, then c may be designated the *defining node* of l and all other concepts in l are called *bound nodes* of l.
- A line of identity may consist of a single concept c, which is the defining node with no bound nodes.

3.1.5 Graphical notation

So far we have been displaying conceptual graphs using the graphical representation. Figure 3.4 shows an example with a couple of levels of nesting.

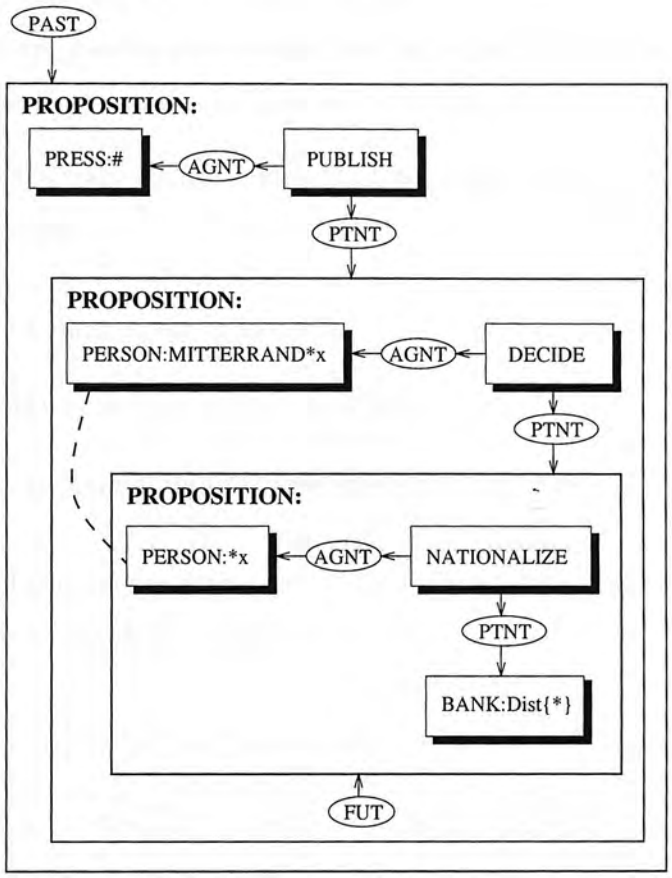


Figure 3.4: Graphical representation

3.1.6 Linear notation

In order to represent conceptual graphs in textual form a linear notation is used. The concept [TYPE: Referent] is represented as `[TYPE: Referent]`. Relations are written: `->(REL)->` or `<-(REL)<-`. If a concept is linked to more than two relations then the relations can be written on new lines. In such cases a hyphen is used to indicate that the relations are continued on subsequent lines.

```
[CONCEPT]-
   ->(REL1)-> ...
   <-(REL2)<- ...
```

Drawing identity lines for coreferential concepts is not possible in text mode. That is why variables are used in the referent field of a concept (`[TYPE: *var]`). If two concepts have the same variable in their referent fields they are the same concept.

There are some arbitrary choices which must be taken when encoding a conceptual graph in a linear form:

- which node to pick as the head
- which relation to list first in the linear form
- which node to mark with a variable when breaking cycles

Conceptual graphs terminate with a dot (“.”). Figures 3.5 and 3.6 show a conceptual graph written in the graphical and linear notations.

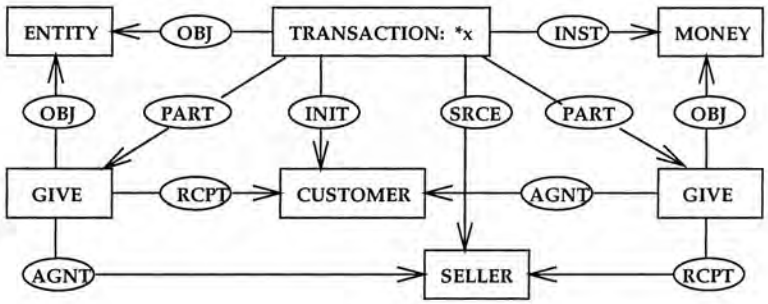


Figure 3.5: A more complex conceptual graph

3.1.7 Graph Unification

This section provides the definition of graph unification (maximal join on two graphs). As maximal join is not a primitive operation, the operations that it is built on are first defined (join, join on compatible projections). Definitions of the structures involved in the maximal join operation like compatible projections and maximally extended compatible projections are also given.

[3] The final punctuation indicates the end of the relations list for `[GIVE]` (first comma), then the end of the relations list for `[TRANSACTION]` (second comma) and the end of the conceptual graph (period).

```
[TRANSACTION]-
  (INIT)->[CUSTOMER: *customer]
  (SRCE)->[SELLER:   *seller  ]
  (OBJ)->[ENTITY: *entity]
  (INST)->[MONEY: *money]
  (PART)->[GIVE]-
             (AGNT)->[CUSTOMER: *customer]
             (RCPT)->[SELLER:   *seller  ]
             (OBJ)->[MONEY: *money],
  (PART)->[GIVE]-
             (AGNT)->[SELLER:   *seller  ]
             (RCPT)->[CUSTOMER: *customer]
             (OBJ)->[ENTITY: *entity],      ,.
```

Figure 3.6: A graph in the linear notation[3]

Definition: *Join*

A join between two graphs u and v on compatible concepts c and d is a graph w obtained by: taking the two graphs u and v and replacing c by $c \cap d$ (maximal common specialisation of c and d), deleting d, and linking all relations that had been linked to d to $c \cap d$.

```
u:                [PERSON: Sue]<-(AGNT)-[EAT].
                  ^^^^^^^^^^^^^

v: [EAT]-(AGNT)->[GIRL].
                  ^^^^^^

w: [EAT]-(AGNT)->[GIRL: Sue]<-(AGNT)-[EAT].
```

An extreme case of joining two graphs on two concepts is when one of the graphs consists of just one concept. The result then is the other graph with the corresponding concept restricted.

Definition: *Specialisation*

A graph u is a specialisation of a graph v ($u \leq v$) if:

1. u and v are the same graph;

2. u introduces a coreference link between two compatible concepts in v; or

3. u is the join of a specialisation of v with another graph.

Definition: *Generalisation*

A graph v is a generalisation of a graph u ($v \geq u$) if u is a specialisation of v.

A generalisation hierarchy on graphs can also be defined. The graph $\boxed{\top}$ is the most general graph. Common generalisation and common specialisation of two graphs are defined trivially.

Definition: *Projection*

A projection of a graph v onto a graph u which is a specialisation of v ($u \leq v$), is a specialisation u' of v embedded in u s.t. u' does not introduce new nodes for v. π is the projection operator: $u' = \pi v$.

```
v : [PERSON   ]<-(AGNT)-[ACTION].
u : [GIRL: Sue]<-(AGNT)-[EAT   ]-(MANR)->[FAST].
u': [GIRL: Sue]<-(AGNT)-[EAT]
```

The projection of graph on a specialisation is not necessarily unique. Also for two different concepts or relations $x_1 \neq x_2$ it might happen that $\pi x_1 = \pi x_2$.

Definition: *Compatible projections*

If two conceptual graphs u_1 and u_2 have a common generalisation v then the projections $\pi_1 : v \rightarrow u_1$ and $\pi_2 : v \rightarrow u_2$ are compatible if $\pi_1 v$ and $\pi_2 v$ have a common specialisation w_i.

```
u1: [GIRL       ]<-(AGNT)-[EAT   ]-(MANR)->[FAST].
u2: [PERSON: Sue]<-(AGNT)-[EAT   ]-(OBJ)->[PIE].
v1: [PERSON     ].
w1: [GIRL:   Sue].
v2:                       [ACTION].
w2:                       [EAT   ].
v3: [PERSON     ]<-(AGNT)-[ACTION].
w3: [GIRL:   Sue]<-(AGNT)-[EAT   ].
```

Compatible projections for the generalisation v_1 of u_1 and u_2 are:

$\pi'_1 v =$ [GIRL] , and

$\pi'_2 v =$ [PERSON: Sue].

Compatible projections for the generalisation v_3 of u_1 and u_2 are:

$\pi_1 v =$ [GIRL]<-(AGNT)-[EAT] , and

$\pi_2 v =$ [PERSON: Sue]<-(AGNT)-[EAT].

Definition: *Join on compatible projections*

If two conceptual graphs u_1 and u_2 have a common generalisation v with compatible projections $\pi_1 : v \rightarrow u_1$ and $\pi_2 : v \rightarrow u_2$ then the join between u_1 and u_2 on these compatible prejections is the most general graph w s.t.

1. w is a common specialisation of u_1 and u_2 with projections $\pi'_1 : u_1 \rightarrow w$ and $\pi'_2 : u_2 \rightarrow w$; and
2. $\pi'_1 \pi_1 v = \pi'_2 \pi_2 v$.

Definition: *Maximally extended compatible projections*

If two graphs contain compatible projections of a common generalisation v, those projections might be extended by finding a larger common generalisation that includes v as a subgraph. Since all conceptual graphs are finite, the process of extension must eventually stop. The resulting compatible projections are called maximally extended.

Finding maximally extended compatible projections of two graphs is a non-deterministic operation!

Definition: *Maximal Join*

A maximal join between graphs u and v is a join on a maximally extended compatible projection. [Sowa 84, p.103]

The graph w below is the maximal join of the graphs $u1$ and $u2$. In this case there is only one maximal join between $u1$ and $u2$.

```
u1: [GIRL        ]<-(AGNT)-[EAT   ]-(MANR)->[FAST].
u2: [PERSON: Sue]<-(AGNT)-[EAT   ]-(OBJ)->[PIE].
w : [GIRL:   Sue]<-(AGNT)-[EAT   ]-
                                   (OBJ)->[PIE]
                                   (MANR)->[FAST],.
```

The maximal join operation is based on finding maximally extended compatible projections which is a non-deterministic operation. Hence, maximal join is also non-deterministic!

Maximal join on two graphs can fail if two graphs do not have *any* compatible concepts. This might be too weak in certain cases and in order to restrict the maximal join operation a measure of how good the matching was might have to be introduced.

3.1.7.1 Maximal Projections and a Maximal Subgraph

A crucial part of the maximal join operation is finding maximal projections in both graphs. This step might be reminiscent of the problem of finding a maximal common subgraph in two graphs but there are certain distinctions that have to be made. We first define the operation of finding a maximal common subgraph and then compare it with the operation of finding maximal projections.

Definition: *SubGraph*

A subgraph of a graph $G =< N, E >$ is graph $G' =< N', E' >$ such that: $N' \subseteq N$, and $E' \subseteq < N' \times N' > \cap E$.

As this definition stands it does not preclude a subgraph being not connected.

Definition: *Graph Isomorphism*

Two graphs $G =< N, E >$ and $G' =< N', E' >$ are isomorphic if they have the same structure: i.e., there are bidirectional mappings FN and FE between the nodes and arcs of one graph onto the nodes and arcs of the other ($FN : N \leftrightarrow N'$, $FE : E \leftrightarrow E'$) such that:

$(FN : node_1 \leftrightarrow n_1, FN : node_2 \leftrightarrow n_2$, and $FE : arc(node_1, node_2) \leftrightarrow a(x, y)) \Rightarrow$
$(x = n_1$ and $y = n_2)$

Definition: *Common SubGraph*

A common subgraph of two graphs is a subgraph of the first graph that is isomorphic with a subgraph of the second graph.

The notion of a *maximal common subgraph* is defined in terms of an ordering relation that tells if one subgraph is better than another. Often this ordering relation depends on the particular application and in some cases the ordering relation could take into account the number of nodes or number of relations or both. [McGregor 82] describes a backtrack search algorithm for finding the maximal common subgraph which considers a maximal common subgraph to be one with the most arcs in it.

Finding a maximal common subgraph is problem of the type: "Find a best solution", thus the operation is a deterministic one. However the search space to be explored is different from the search space of all maximal projections—it is bigger.

In order to see the differences between finding the maximal common subgraph and maximal projections consider Figure 3.7.

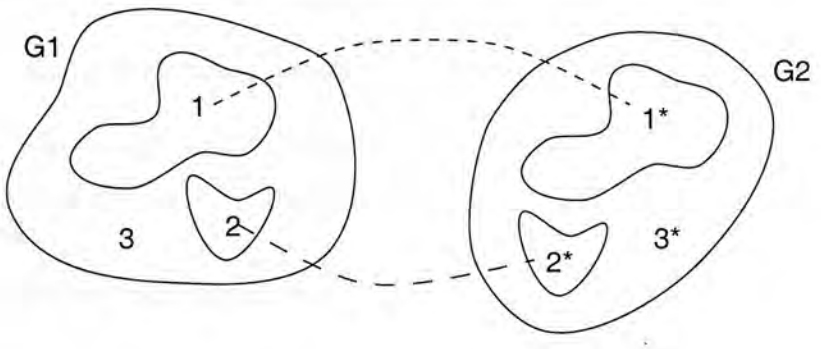


Figure 3.7: Maximal common subgraph vs. maximal join

Let graph $G1$ be represented as three regions 1, 2 and 3 while graph $G2$ consists of $1*$, $2*$ and $3*$. If the regions 1 and $1*$ are isomorphic and so are 2 and $2*$ then as maximal projections will be 1 corresponding to $1*$ and 2 corresponding to $2*$. In order to find the maximal common subgraph the case where 1 and 2 taken together correspond to

1* and 2* (again viewed as one graph) has to be considered. And usually that would be the preferred solution.

If there are n connected corresponding subgraphs there will be n solutions for the maximal projections while the space to be explored by the common maximal subgraph operation will consist of $2^n - 1$ states.

This difference stems from the fact that the maximal common subgraph can be disconnected. If a subgraph (and a graph) is restricted to be connected then the search spaces for the maximal common subgraph and maximal projection operations are the same.

In the following we will be using matching of two graphs to refer to performing a maximal join on them. Matching will be used in the following cases:

1. matching the initial semantics against the (applicability) semantics of a mapping rule

2. matching the corresponding semantics to complements of a mapping rule to get the corresponding semantics of the whole construction (which might be different from the input semantics)

We will discuss these at greater length in Chapter 5.

Warning: Matching is used to mean: checking whether two graphs can be matched and the result of the matching operation on two graphs!

3.1.8 Defining new concepts

To define new concepts lambda abstractions are used:

```
type BLACK-DOG(x) is
        [DOG: *x]->(ATTR)->[BLACK].
```

Conceptual relations can also be defined in a similar fashion.

3.1.9 Canonical formation rules

Canonical graphs have the same format as conceptual graphs. They define selectional restrictions/constraints on the way conceptual graphs and relations can be put together. The fundamental distinction between canonical formation rules and inference rules is that inference rules always derive true graphs from true graphs while canonical formation rules give possible conceptual graphs that we can talk about.

In our presentation we omit the rules of inference as this would lead us to a large topic of automated reasoning with conceptual graphs which we do not employ for the purposes of the sentence generation techniques described in this thesis.

SUMMARY

- ⊙ CGs are one of the most formalised approaches among the semantic network frameworks.
- ⊙ CGs are a formally defined, mathematical abstraction. They are expressive and can be used to model NL semantics.
- ⊙ CGs have been extensively studied from a computational point of view and efficient matching algorithms for them have been developed.
- ⊙ CGs can be expressed in different notations (including graphically) and the display of CGs can be tailored according to the application.
- ⊙ CGs are used in a variety of applications.

Chapter 4

Non-Concatenative Grammars

This chapter discusses the syntactic representations that our generator produces. We briefly mention Context-Free Grammars and give an example of why a larger domain of locality than that provided by CFGs might be needed. We then discuss more sophisticated approaches to syntactic theories—the class of the so-called non-concatenative grammars. We introduce one of the main representatives of this class — Tree Adjoining Grammars which was the basis of our initial generation system (PROTECTOR-95, [Nicolov *et al.* 95]). We then discuss a new syntactic framework — D-Tree Grammars which are descendants of Tree Adjoining Grammars and which are more suitable for generation. D-Tree Grammars are the syntactic backbone of our generation framework. As the system described in this thesis has pioneered generation with D-Tree Grammars we make the effort to describe them in detail so as to allow replication of our results.

4.1 Introduction

4.1.1 Context-free grammars

If the idea of templates (which we discussed in the previous chapter) is pushed even further two issues become of paramount importance: (i) being able to dismantle the input representation; (ii) being able to have deeper embedded templates which group constituents as a proper linguistic theory would suggest; and (iii) being able to keep correspondences between (parts of) the input representation and the embedded invocation of templates.

Context-free grammars (CFGs) are a formalism that allows one to address these issues and there are a number of generation approaches using context-free based syntactic formalisms like GPSG [Busemann 89], LFG [Kohl 91], HPSG. CFGs are not the topic of this chapter or thesis and we will not introduce them.[1]

Generation systems that use linguistically motivated syntactic representations have the following advantages: they are easier to maintain; they allow for more sophisticated higher-quality texts; they can be made to produce simultaneous multilingual text (in a much more principled way than UNIX error messages for example) and made to conform to certain document standards of sublanguage.

CFGs however, make the syntactic structures too fragmented. For example, while sentences consist of subject group NP and a predicate group VP an individual verb imposes constraints on its objects (case) as well as its subject (person, number, case: nom) yet the subject is not in the VP → V... rule. In order for a CFG to relate the subject information (that the verb imposes) to the subject NP node this information has to be propagated through a mechanism of parameter passing (variable sharing) so that the subject can be in the same CFG production. We will refer to the structures over which we can state constraints as a domain of locality.

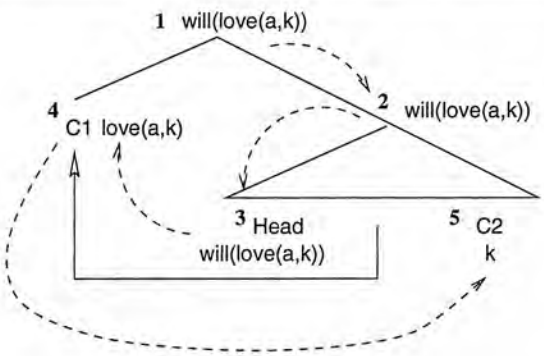


Figure 4.1: Complement displacement

[1] Good introductions to CFGs can be found in [Gazdar & Mellish 89, chapter 4] and from a formal language theory perspective in: [Harrison 78, Hopcroft & Ullman 79, Aho *et al.* 86].

4.1.2 Motivation for larger domain of locality

[Hinrichs *et al.* 94] present a linguitically motivated analysis of partial VP topicalisation which poses problems for CFG-based approaches (even for SHDG). Figure 4.1 shows how the semantics is perculated from node to node. From node 1 we go to the semantic head (node 2) and from there again we visit the semantic head (node 3). At this point however the semantics of node 5 which is the sister of node 3 is not instantiated because this semantics is mediated by the semantics at node 4. Note that it it would not have been possible to visit directly node 4 from node 1 as it is not the semantic head and at the point when we are at node 1 the semantics at node 4 is uninstantiated. Figure 4.2 gives the actual HPSG analysis. This example is from [Minnen *et al.* 95].

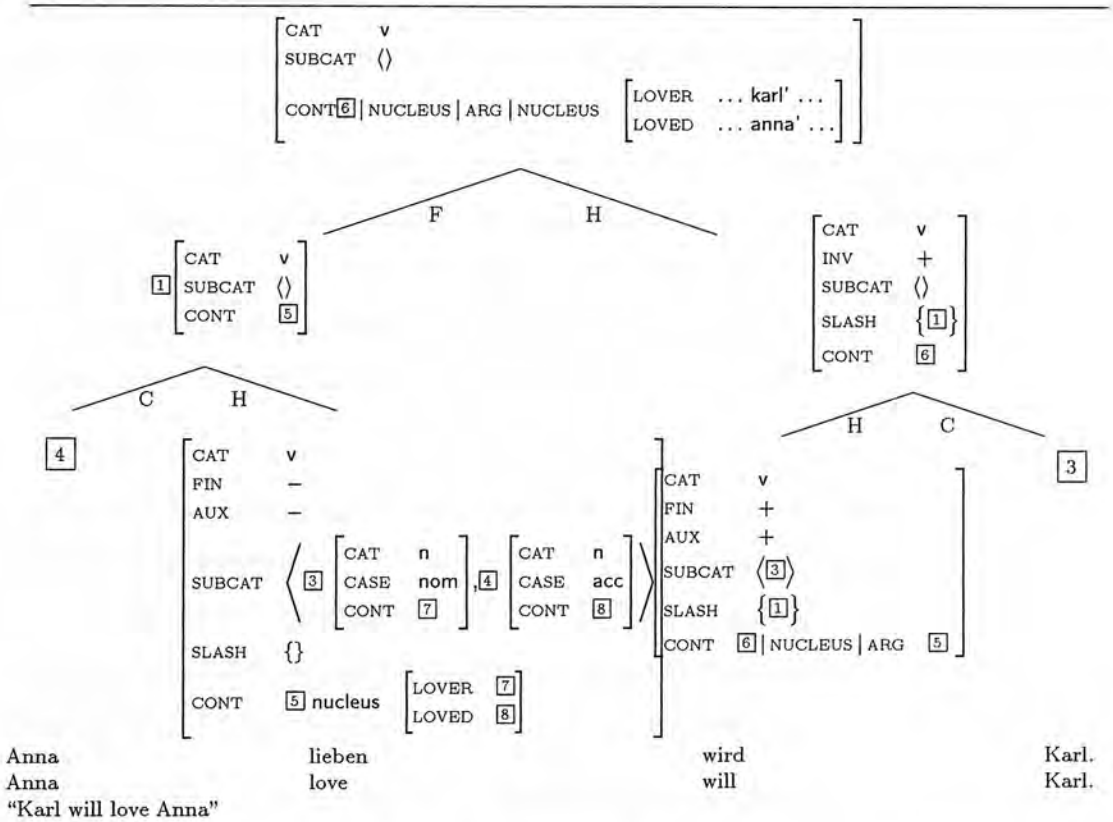


Figure 4.2: Example of problematic complement displacement

The problem is that the semantic head does not directly instantiate the semantics of all its siblings. In the case where a constituent which is not the semantic head but which determines the semantics of another constituent there will be a problem to do with the

impossibility for a generator to decide in what order to visit the RHS constituents. As this example shows there is no such order over the CFG skeleton. However, if we had a larger structure in which we could locally state the order of visiting nodes (larger domain of locality) then we would be able to resolve this issue.

In the remainder of this chapter we will look at grammars with larger domain of locality.

4.1.3 Non-concatenative grammars

Current constraint-based formalisms in computational linguistics assume that syntactic phrases are built up by concatenation (hence the use of a context-free backbone). The value of the string at the mother node is the concatenation of the strings of the daughter nodes. This assumption is sometimes challenged allowing more powerful operations to construct strings. Often the linguistic motivation for such alternative conceptions of string combination are analyses of discontinuous constituency (extraction phenomena). From a formal language theory point of view non-concatenative formalisms are more powerful in the sense that their corresponding languages properly include the context-free languages. From a semantic perspective such formalisms can facilitate a systematic, compositional construction of semantic structures.

There are a number of formalisms that have been proposed in the literature that use additional machinery beyond concatenation: Head-Wrapping grammars [Pollard 84], Mark Johnson's use of the 'combine' operator in the analysis of the Australian free word order language Guugu Yimidhirr [Johnson 85], Mike Reape's use of 'sequence union' in the analysis of Germanic semi-free word order constructions [Reape 89, Reape 90]. The categories in Categorial Grammars also have a larger domain of locality [Wood 93].

We will look at two closely related representatives of non-concatenative grammars: Tree-Adjoining Grammars and D-Tree Grammars.

4.2 Tree-Adjoining Grammars (TAGs)

This section introduces Tree Adjoining Grammar. The historical evolution of the formalism up to the present moment is shown and illustrative examples are given

assuming the latest version of Tree Adjoining Grammar.

A Tree Adjoining Grammar (TAG) is a grammatical formalism that associates syntactic trees with a list of words representing a sentence in a language. Tree Adjoining Grammars were introduced in [Joshi *et al.* 75]. The first study of this system, from the point of view of its formal properties and linguistic applicability, was carried out in [Joshi 85]. A detailed study of the linguistic relevance of TAG was reported in [Kroch & Joshi 85].

In the context of generation, TAGs have been used in a number of systems: MUMBLE [McDonald & Pustejovsky 85, Meteer *et al.* 87], SPOKESMAN [Meteer 90], WIP [Wahlster *et al.* 91, Harbusch *et al.* 91], the system reported in [McCoy *et al.* 92], the first version of PROTECTOR [Nicolov *et al.* 95], SPUD [Stone & Doran 97], FLAUBERT [Danlos & Meunier 96] and VMGeCo [Becker *et al.* 98]. In the area of grammar development TAGs have been the basis of one of the largest grammars developed for English—XTAG [Doran *et al.* 94]. It is important to draw the distinction between TAG as a (mathematical) formalism and TAG as a particular grammar. The TAG formalism plays the same role as CFG schematic productions for the rewriting of typed feature structures in HPSG.

TAG is attractive for sentence generation because [Joshi 87]:

1. On the syntactic level TAG trees factor out recursion of syntactic structures (in the auxiliary trees) and localise dependencies such as subcategorisation and filler-gap (in the initial trees). This makes it easy to describe syntactic phenomena.

2. TAG trees capture the function argument structure. This simplifies (in comparison to the CFG case) the mapping at the semantics-syntax interface level and allows for incremental processing.

3. TAGs can be lexicalised. For parsing once we know the set of trees that can be anchored by every word in the input we do not need to consider any other trees in the grammar. Similarly for generation if we know the trees that can be anchored by lexical items that match the input semantics we only need to consider the combinations of these trees (which is of course constrained by the semantics) and not any other trees in the grammar.

4.2.1 The TAG formalism

A Tree Adjoining Grammar is specified by a finite set of *elementary* trees. The elementary trees correspond to *minimal* linguistic structures that localise dependencies such as subcategorisation and filler-gap. The non-frontier nodes of the trees are labelled with grammatical categories. The leaf (frontier) nodes in elementary trees can be labelled with lexemes (terminal symbols) or (like non-terminal nodes) with grammatical categories in the usual way for (incomplete) syntactic trees. Elementary trees are divided into *initial trees* and *auxiliary trees.*

Initial trees represent minimal sentence structures (the root of an initial tree is always an S node). The frontier nodes of an initial tree are labelled with preterminal lexical category symbols: N, V, A, P, DET, etc.[2]

Auxiliary trees correspond to minimal recursive structures. Thus, if the root node of an auxiliary tree is labelled by a non-terminal symbol X then there is a distinguished node (called the *foot node*) in the frontier of this auxiliary tree which is also labelled by X. Foot nodes in auxiliary trees are marked with $*$ (e.g., X$*$). The rest of the frontier nodes are preterminals.[3] Thus we cannot have auxiliary trees that look like:

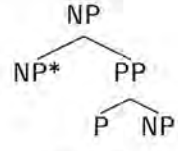


Figure 4.3: Non-auxiliary tree in the original pure TAG

Auxiliary trees correspond to modifiers or to predicates (verbs) taking sentential complements.

Auxiliary and initial trees are distinguished by the presence (or absence, respectively) of a foot note. It is not possible for a tree to be simultaneously an auxiliary tree and an initial tree.

[2] From a mathematical perspective initial trees have all their frontier nodes labelled with terminal symbols—all possibilities are multiplied out.

[3] Again from a formal point of view no other frontier nodes other than the foot is a nonterminal. The rest are terminals.

4.2.1.1 Adjunction

In TAG bigger trees are composed by *adjoining* an auxiliary tree into another tree which is either an initial tree or the result of a previous adjoining operation. Adjoining can be described as expanding a non-terminal node X in a tree (adjunction node) into an auxiliary tree (with a root and foot nodes labelled with X) and attaching the subtree that was originally rooted in the adjunction node X to the foot of the auxiliary tree. See Figure 4.4. In the picture we show the two attachings of trees and specify a certain order in which these operations are carried out but they can be be done in the other order.[4]

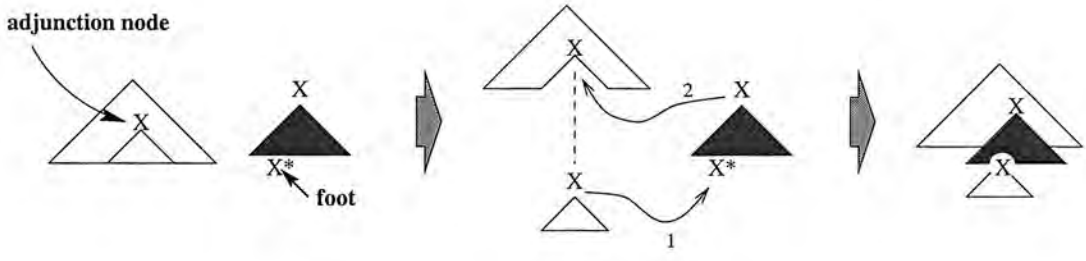


Figure 4.4: The adjoining operation

Figure 4.5 gives a linguistic example of adjunction:

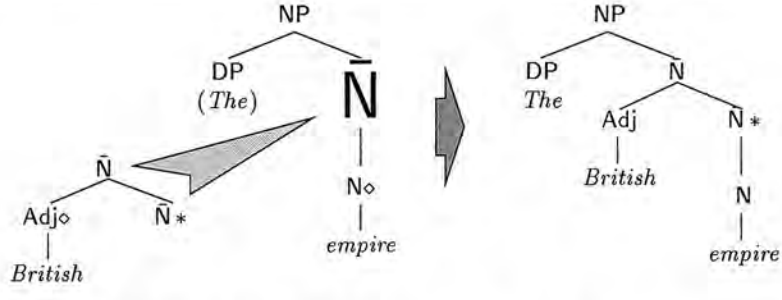


Figure 4.5: A linguistic example of adjunction

[4] The order does not matter because we check for equality of the non-terminal symbols. Most systems use labels for nodes that are feature structures (see Section 4.2.3) and there again the order of carrying out the attachments does not matter due to the associativity of unification.

Formally a TAG is a five-tuple $\langle \sigma, N, I, A, S \rangle$ where:

σ is a finite set of *terminal symbols*
N is a finite set of *non-terminal symbols*
I is a finite set of *elementary initial trees*
A is a finite set of *elementary auxiliary trees*
S is a distinguished non-terminal, the *start symbol*

Adjunction in effect encodes string wrapping and is thus more powerful than concatenation. Adjunction generates languages and trees which cannot be generated by the use of substitution (and thus by CFGs).

In order to be linguistically meaningful, the elementary trees in a TAG must conform to certain constraints that are not explicitly expressed in the definition of the formalism. In particular, each elementary tree must constitute a *minimal linguistic structure* elaborated up to preterminal (terminal) symbols[5] and containing a head and all its complements or a modifier. Initial trees have essentially the structure of simple sentences; auxiliary trees correspond to minimal recursive constructions and generally constitute structures that act as modifiers of the category appearing at their root and foot nodes.

A hypothesis that underlies the linguistic intuitions of TAGs is that all dependencies are captured within elementary trees. This is based on the assumption that elementary trees are the appropriate domain upon which to define dependencies, rather than, for example, productions in CFG. Since in string-rewriting systems dependent lexical items cannot always appear in the same production, the formalism does not prevent the possibility that it may be necessary to perform an unbounded amount of computation in order to check that two lexical items agree in certain features. However, since in TAGs dependencies are captured by bounded structures, we expect that the complexity of this computation does not depend on the derivation. Features such as agreement may be checked within the elementary trees (instantiated up to lexical items) without the need to percolate information up the derivation tree in an unbounded way. Some checking is necessary between an elementary tree and an auxiliary tree adjoined to it at some node, but this checking is still local and bounded. Similarly, elementary trees, being minimal linguistic structures, capture all of the subcategorisation information.

[5] In the original TAG one is not allowed to have an NP node in the frontier.

4.2.2 Lexicalised Tree-Adjoining Grammar (LTAG)

Lexicalised Tree-Adjoining Grammar (LTAG) introduces some new machinery. Elementary trees are now associated with lexical items and a new operation is introduced.

Initial trees are more general—they no longer correspond only to minimal sentential structures but to minimal linguistic structures (including NPs). They now have to have at least one lexeme at the frontier which is in a way central for the local structure of the initial tree and is called the *anchor*. Initial trees are now viewed as describing the arguments of the anchor. The preterminal node dominating the anchor is marked with $\diamond$ (e.g., A$\diamond$) to indicate the anchor position because often in order to capture certain generalisations the actual anchor is omitted. All the non-terminals at the frontier of an initial tree are to be associated with a subtree by means of a special operation called *substitution*.

As for auxiliary trees if they have other non-terminals than the foot node they have to marked for "expansion" by means of (again) the substitution operation.

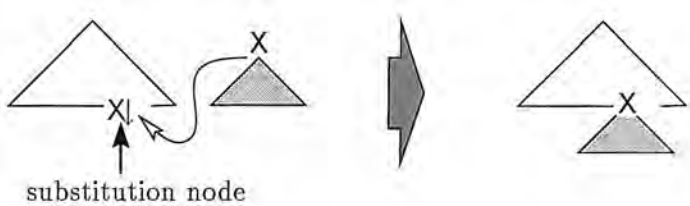


Figure 4.6: The substitution operation

4.2.2.1 Substitution

The additional operation used in lexicalised TAG (substitution) works as follows. A tree can substitute (or replace) a frontier non-terminal node in another tree. The root node of the substitution tree and the frontier node have to be labelled with the same category. A node at which substitution is allowed is marked with $\downarrow$ (e.g., X$\downarrow$). By definition a substitution node in auxiliary or initial tree is a frontier node labelled with a non-terminal symbol (and in the case of auxiliary trees is not the foot node). The substitution operation is presented graphically in Figure 4.6.

It is interesting to note that the adjoining operation can be viewed as two successive substitutions as shown in Figure 4.4.

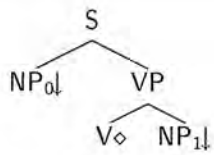


Figure 4.7: Transitive verbs

Figure 4.7 shows an example initial tree for a transitive verb construction. The NPs are numbered so that they can be unambiguously referred to.[6] NP_0 and NP_1 are marked for substitution and V will dominate the anchor.

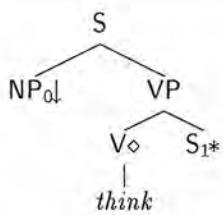


Figure 4.8: Auxiliary tree

Figure 4.8 shows an example auxiliary tree. This is the construction of the verb *think* which takes a sentential complement. The node NP_0 is marked for substitution. The node S_1 is marked as the foot node of the auxiliary tree.

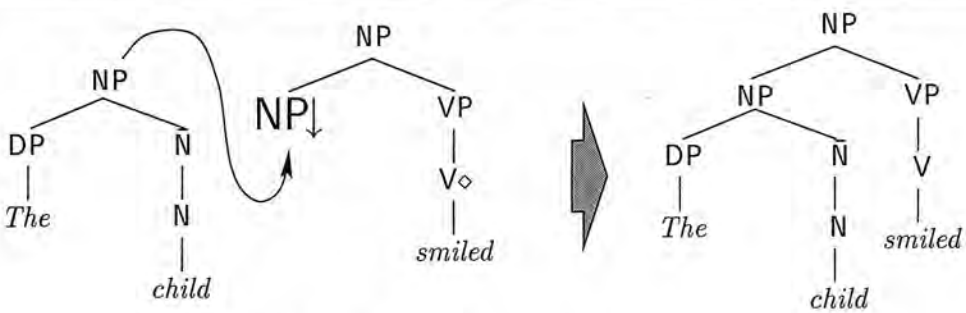


Figure 4.9: A linguistic example of substitution

Figure 4.9 gives a linguistic example of substitution.

[6] Another way of unambiguously referring to a node is to use its elementary tree address. Assume a uniform labeling of the edges between a mother and the daughter nodes (e.g., 1, 2,... for the first, second,... constituents). Then the elementary tree address of a node is the sequence of labels along the path from the root of the tree to the node. The elementary tree address of the root is the empty sequence.

The TAG machinery is further elaborated to include *adjoining constraints* associated with nodes in the trees. In standard TAG each node in a tree was associated with a list of auxiliary trees that can be adjoined at that node. These are called *selective adjoining* constraints. A special case of selective adjoining constraints is the *null adjoining* constraint which specifies that no auxiliary tree can be adjoined at the particular node. *Obligatory adjoining constraints*, on the other hand, specify a list of auxiliary trees one of which has to be adjoined at the particular node in order for the tree to satisfy well-formedness constraints.[7]

4.2.3 TAG within the unification framework (FTAG)

In pretty much the same way as PATR-II non-terminals are represented as feature structures [Shieber 86], elementary trees in TAG can be viewed as specifying a skeleton tree structures where the non-terminals are again feature structures.

4.2.3.1 Adjoining and substitution with unification

If the nodes in elementary trees are feature structures then the adjoining and substitution operations have to be modified appropriately.

It is convenient to adopt two features associated with an adjunction node. The `top` feature can be considered as constraints on that node that are made on the basis of information contained in ancestors and siblings, i.e., this is information corresponding to a view to the top of the non-auxiliary tree from the adjunction node. The `bottom` feature, conversely, represents constraints to do with the tree attached to the adjunction node. It corresponds to a view to the bottom of the tree.

Substitution nodes only have a `top` feature associated with them.

The updating of the feature structures is shown in Figure 4.10 and Figure 4.11 (the auxiliary tree and the substitution tree are shaded).

The adjunction node A, the root node of the auxiliary tree A' and the foot node of the auxiliary tree $A*$ have a `top` and a `bottom` feature associated with the feature

[7] LTAGs can also lexicalise CFGs, i.e., a CFG can be rewritten in a lexicalised TAG (i.e., LTAG) strongly equivalent to the original CFG (preserves the original trees) [Joshi & Schabes 92]

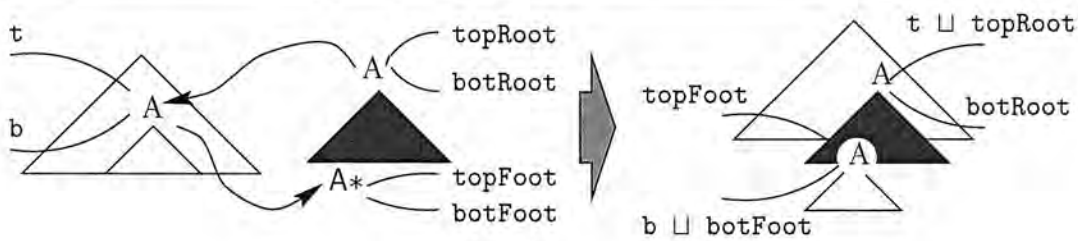


Figure 4.10: Adjunction with unification

structure representing the nodes. We have labelled them t and b (for the adjunction node), topRoot and botRoot (for the root of the auxiliary tree), and topFoot and BotFoot (for the foot of the auxiliary tree). In the resulting tree the top feature of the adjunction node is unified with the top feature of the root of the auxiliary tree. Analogously, the bottom feature of the adjunction node is unified with the bottom feature of the foot of the auxiliary tree. In a way the adjunction node gets split.

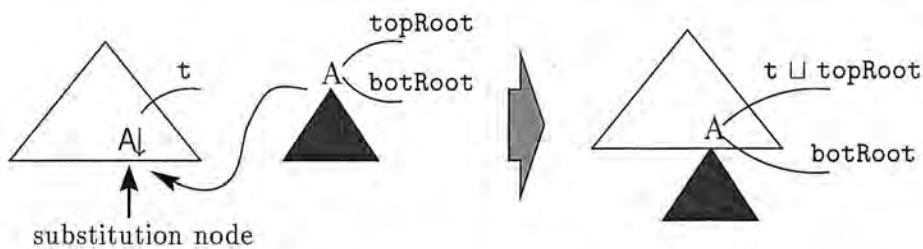


Figure 4.11: Substitution with unification

The substitution operation unifies the top feature of the substitution node (which does not have a bottom feature) with the top feature of the root node of the substitution tree.

Both adjunction and substitution operations are monotonic as far as the feature structures are concerned but all current implementations explicitly build a new tree corresponding to the result of adjunction (see for example [vanNoord 93, page 136]). The operations are thus not monotonic on the representations (dominance information in the original trees is not preserved).[8]

It is interesting to note that if trees with nodes represented by feature structures are used then the adjoining and substitution constraints can be specified in virtue of the

[8] We return to this issue when we discuss representations (descriptions of trees) for DTGs in Section 4.3.8.

information present in the nodes relevant for the operation: the adjunction node and the root and foot nodes of the auxiliary tree in case of adjunction and the substitution node and the root of the substitution tree in case of substitution—if the unifications required by the operation succeed then the particular operation is allowed. This, of course, is natural but the point here is that no exhaustive enumeration of what trees can be adjoined or substituted is needed. Thus, the adjoining and substitution operations can be restricted (constrained) by using appropriate features and values.

When the derivation is completed the `top` and `bottom` features of all adjunction nodes have to be unified. If they do not unify there is an obligatory adjoining constraint. Then an auxiliary tree must be adjoined at that adjunction node. Because as a result of the adjunction operation the adjunction node is split then the incompatibility between its `top` and `bottom` features might be avoided. This operation is called closing off a derivation. For an example of how this mechanism works see Figure 4.16.

4.2.4 Descriptions of trees

If the structures manipulated by the grammar are trees (as we have described them up to now) then, in the process of derivation, we start with an initial tree (trees) and the final result (which is again a tree) preserves only but a part of the information that was originally present in the starting structures. When adjunction is performed at a node, the distance (in terms of the number of immediate dominance links) between the nodes above the adjoining node and those below it is increased. So, certain information is changed as a result of the adjoining operation. The adjunction as described above is thus a non-monotonic operation. The fact that the adjoining operation does not preserve all the the structural relations specified in the trees being composed prevents us from embedding TAG in a unification-based framework. This is so because the unification operation is monotonic. However, this can be remedied by interpreting the structures manipulated by the adjoining operation not as trees but as descriptions of trees. In case of an adjunction node, rather than stating that it immediately dominates another node, the relation can be relaxed to be one of dominance and if the nodes are to be further separated (by adjoining an auxiliary tree between them) then the relation of dominance will still hold.

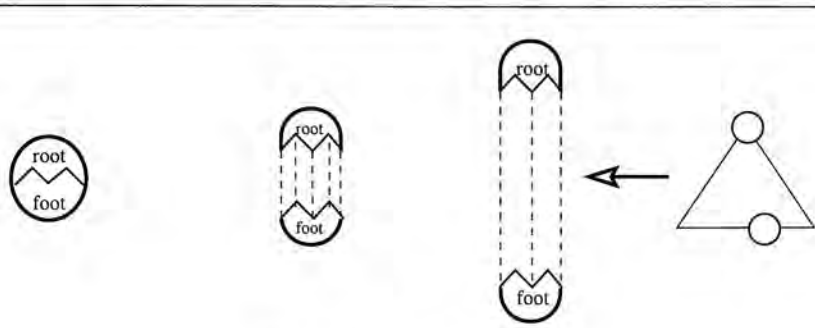


Figure 4.12: Quasi node

One way of translating an elementary tree into a description of a tree is to substitute a pair of nodes which are in a relation of dominance for each adjunction node. Here the dominance relation is reflexive (the pair of nodes can actually be a single node, i.e., they can be merged), transitive and antisymmetric. Such new nodes are called *quasi-nodes* and the corresponding tree descriptions—quasi trees. Within the quasi-pair of nodes the top node is referred to as *quasi-root* and the bottom node as *quasi-foot*.[9]

4.2.4.1 Adjunction and substitution with quasi-trees

Adjunction and substitution based on quasi-trees are very similar to the equivalent operations in FTAG where the adjunction nodes have `top` and `bottom` features associated with them. The extension (in the case of the new interpretation of the objects manipulated by the grammar) is that the `top` feature structure of an FTAG adjunction node is viewed as the quasi-root of a quasi-pair. The `bottom` feature structure then is the quasi-foot.

In fact FTAGs where the adjunction nodes have `top` and `bottom` features associated with them are a small shift towards TAG using descriptions of trees (quasi-trees).

4.2.5 Examples

In this section we put the TAG machinery into practice. We will see how derivations are performed and how restrictions on the adjoining and substitution operations are enforced. We are assuming a variant of TAG that uses quasi-trees as descriptions of

[9] Quasi-trees as descriptions of TAG trees were introduced in [Vijay-Shanker 92].

the objects that are manipulated by the grammar. The nodes in the quasi trees are represented as feature structures. We will be representing quasi nodes as different tree nodes connected with a dashed arc. The examples are taken from [Vijay-Shanker 92].

Let us consider the derivations of the following sentences:

1. *Who did John see?*
2. *Who did Peter think John saw?*
3. *I wonder who John saw?*
4. *I wonder who Peter thought John saw?*
5. *Peter thought John saw Mary.*

Let the TAG grammar consist of the auxiliary trees in Figure 4.13 and the initial trees in Figure 4.14.

Who did John see? is the result of adjoining the third auxiliary tree (*did*) at the second quasi-pair (S_3,S_4) of the first initial tree. The root of the auxiliary tree after the adjoining will be the quasi-pair:

$$S_3[\text{INV: } \boxed{1}]$$
$$\vdots$$
$$S\,[\text{INV: } +]$$

and the foot of the auxiliary tree after the adjoining (cf. nearest S node dominating *John*) will be as before: S[INV: −] as unifying it with S[INV: −] (the foot of the auxiliary tree) does not add new information.

The resulting tree after the adjoining which corresponds to example (1) *Who did John see?* is shown on the right in Figure 4.15.

The other examples can be constructed in the same fashion. Rather than exhaustively showing how to perform the derivations for all of them we will point out how obligatory adjoining constraints occur as a result of unification and structure sharing.

If quasi-nodes S_1 and S_2 of the first initial tree (α_1e) are unified (the result of the unification will be $S_{1\cup 2}$[wh: +, main: +]) then an obligatory adjoining constraint will

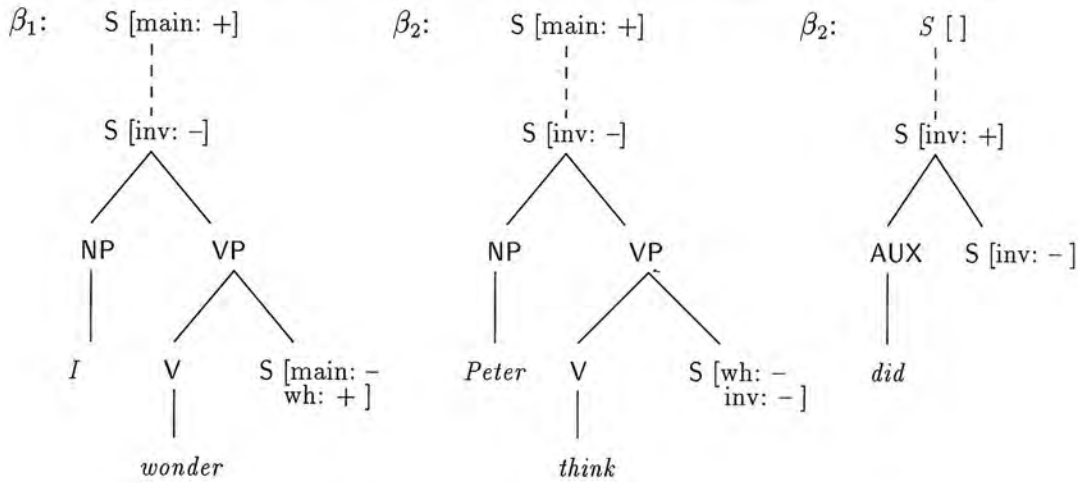


Figure 4.13: Auxiliary trees

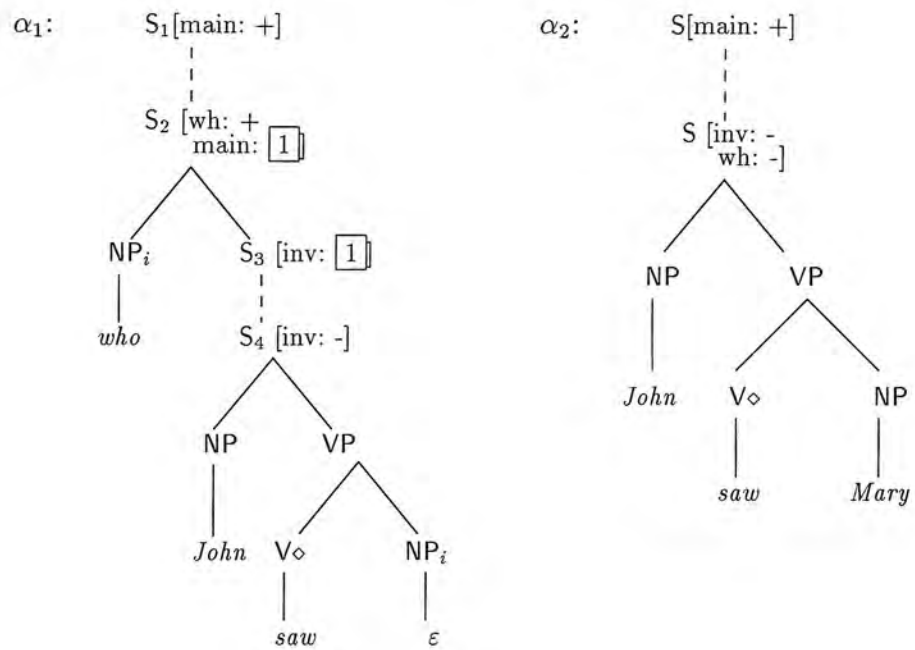


Figure 4.14: Initial trees

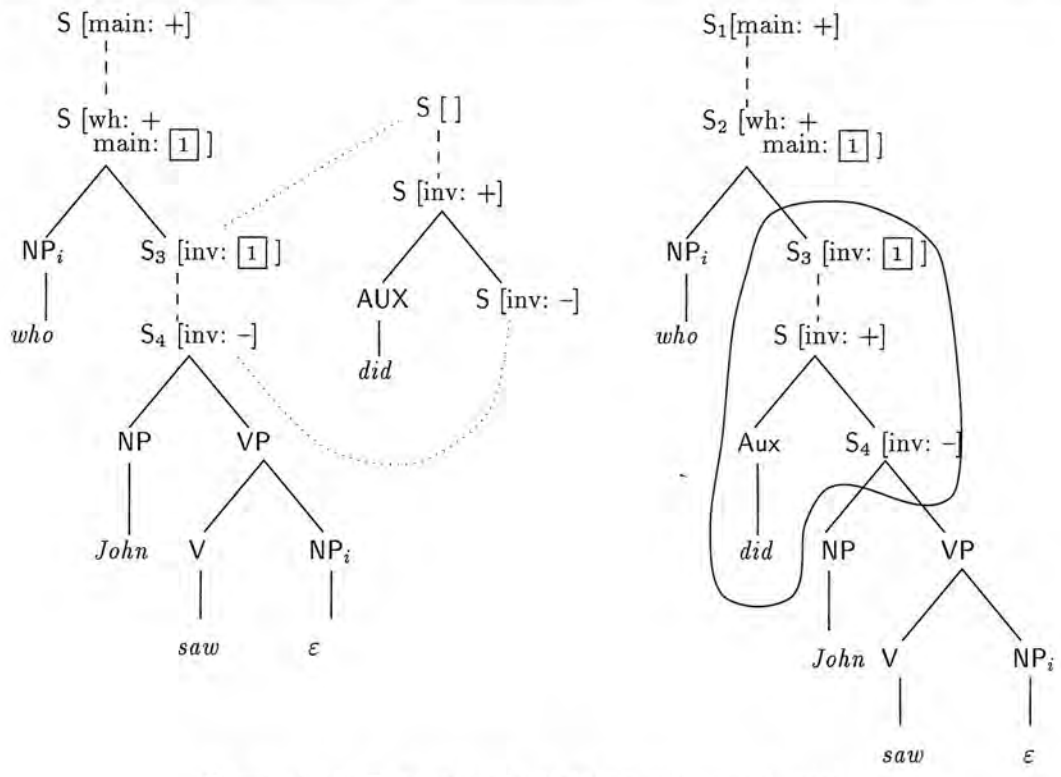


Figure 4.15: Example of adjoining with quasi-trees

appear for the quasi-pair (S_3,S_4) as the feature structures associated with the quasi-root (S[inv: +] the value of the inv feature of S_3 is shared with the inv feature of S_2 which after the unification has become '+') and the quasi-foot (S[inv: −]) will no longer be compatible:

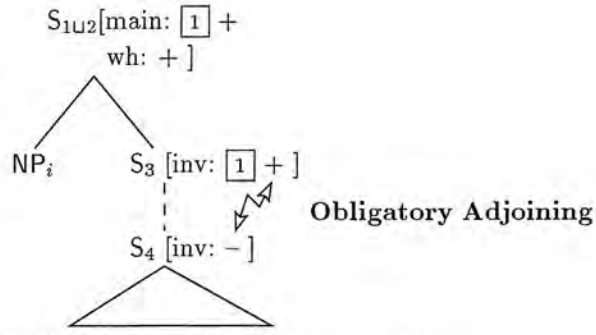


Figure 4.16: Obligatory adjoining constraints

An auxiliary tree (like the third auxiliary tree—*did*) will have to be adjoined at (S_3,S_4) before completing the derivation. This will account for the well-formedness of (1) and (2) above.

Alternatively if quasi-nodes S_3 and S_4 are unified then an obligatory adjoining contraint will appear for the quasi-pair (S_1,S_2). This is the case for sentence (3) above.

TAGs are a tree rewriting system. The *tree set* $\mathcal{T}(G)$ of a given TAG grammar G is the set of all derived trees whose root unifies with the distinguished symbol S (i.e., sentences) and whose frontier nodes are all terminals (all substitution nodes have been filled in). The *string language* $\mathcal{L}(G)$ generated by the TAG G is the set of all terminal strings of trees in $\mathcal{T}(G)$.

4.2.6 Derivations trees vs. derived trees

In the TAG literature the way the derivation proceeds can be described by means of *derivation trees* which are the equivalent to proof trees in logic programming. A derivation tree is a canonical structure that abstracts all possible derivation orders (i.e., ways of combining elementary trees to produce the final derived tree). The nodes of a derivation tree are labelled with names of elementary trees in the grammar. The labels of the domination links in the derivation tree have a type (marking either substitution or adjunction of the tree named at the daughter node) and refer to an elementary tree address in the tree named at the mother node. This is the address (node) at which the operation (substitution or adjunction) takes place. Multiple adjunction (and substitution for that matter) at the same tree address (node) are not allowed.

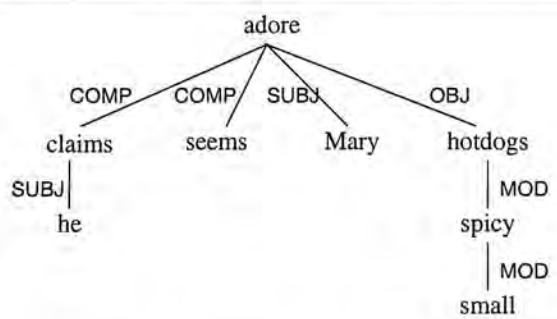


Figure 4.17: TAG derivation

Figure 4.17 gives the derivation tree for the sentence *Small, spicy hotdogs he claims Mary seems to adore.* The SUBJ and OBJ labels of the domination links refer to substitution of the subject and object NPs. COMP (sentential complement) and MOD refer

to adjoining of the daughter trees in the tree named at the mother node. Observe that this derivation structure cannot be directly related to a semantic representation. Both adjectives modify (in a sense independently) *hotdogs* but in the derivation structure one is a daughter of the other (which is a consequence of the surface order). Also the fact that *adore* is an argument of *seem* which in turn is an argument of *claims* is not obvious. Even worse, if we interpret (as one would normally assume) embeddedness $X(Y)$ to mean 'X has as argument Y' in the case of *seem* and *adore* the 'argument-of' relation is reversed. The derivation also relates *adore* and *claims* when neither is an argument of the other. The new grammatical framework that we introduce in the next section addresses these very issues.

The first implementation of our generation system PROTECTOR-95 used TAGs [Nicolov *et al.* 95]. Since then a more recent TAG-related formalism has been proposed, D-Tree Grammars, which we have found to be better suited for generation thanks to a better match of the operations performed on the semantic and syntactic structures. This allowed us to specify the relation between semantics and syntax (which is so important in generation) in a more direct fashion.

4.3 D-Tree Grammars (DTGs)

In this section we discuss D-Tree Grammars (DTGs) and examine their relevance to NLG. DTGs are seen attractive for generation because:

1. DTGs provide a uniform treatment of complementation and modification at the syntactic level.
2. This can lead to simplification of the overall generation architecture.

D-Tree Grammar (DTG) is a new grammar formalism, which arises from work on Tree-Adjoining Grammars [Rambow *et al.* 95a]. TAGs, however, have two limitations which provide the motivation for this work:

1. The TAG operations of substitution and adjunction do not map cleanly onto the relations of complementation and modification.

2. TAGs cannot provide analyses for certain syntactic phenomena:

   (a) long-distance scrambling in German [Becker et al., 1991];

   (b) Romance Clitics [Bleam, 1994];

   (c) wh-extraction out of complex picture-NPs [Kroch, 1987] and

   (d) Kashmiri wh-extraction [Rambow *et al.* 95a].

DTG tries to overcome these problems while remaining faithful to what is seen as the key advantages of TAG (in particular, its enlarged domain of locality).

The details of the remainder of this section (4.3.1—4.3.6) are largely based on [Rambow *et al.* 95a]. The reason why we provide so much detail is to enable replication of our work—there have been only a few publications on DTGs. Our contributions lie in the first implementation of the DTG formalism (including a feature-based version and the use of descriptions of d-trees in the implementation which predates attempts to reformulate DTGs using tree descriptions) and developing the first sizable grammar for English in DTG.

4.3.1 The DTG formalism

DTG (like TAG) assumes the existence of elementary structures. These structures are trees which represent the building blocks for sentences. Rather than working directly with trees we will make use of tree descriptions (called *d-trees* hence the name of the formalism). A description of a tree is a directed acyclic graph with two types of edges: domination edges (*d-edges*) and immediate domination edges (*i-edges*). D-edges and i-edges express domination and immediate domination relations between nodes. These relations are never rescinded when d-trees are composed. Thus, nodes separated by an i-edge will remain in a mother-daughter relationship throughout the derivation, whereas nodes separated by a d-edge can be equated or have a path of any length inserted between them during a derivation.[10] D-edges and i-edges are not distributed arbitrarily in d-trees. For each internal node, either all of its daughters are linked by i-edges or it has a single daughter that is linked to it by a d-edge. Each node in a d-tree

[10] The domination relation is the reflexive transitive closure of the immediate dominance relation.

is labelled either with a terminal symbol, a nonterminal symbol or the empty string ε. A d-tree containing n d-edges can be decomposed into $n + 1$ components containing only i-edges (i.e., proper trees).

In DTG smaller structures are put together to form larger trees by means of two operations—subsertion and sister-adjunction.

4.3.2 Subsertion

When a d-tree α is subserted into another d-tree β, a component of α is substituted at a frontier nonterminal node (a substitution node) of β and all components of α that are above the substituted component are inserted into d-edges above the substituted node or placed above the root node (see Figure 4.18).

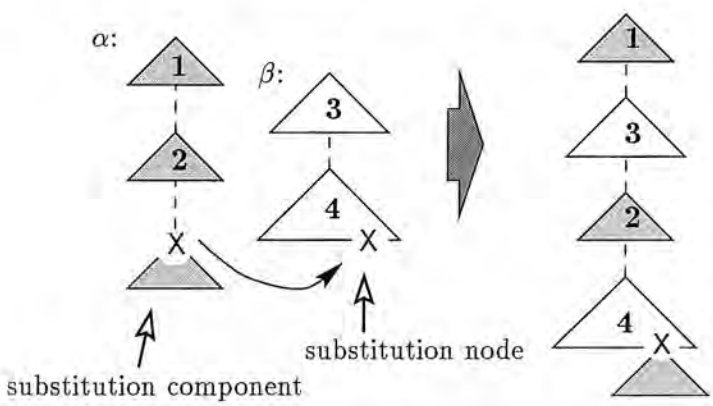


Figure 4.18: Subsertion

In general, when a component $\alpha(i)$ of some d-tree α is inserted into a d-edge between nodes N_1 and N_2 two new d-edges are created, the first of which relates N_1 and the root node of component $\alpha(i)$, and the second of which relates the frontier node of $\alpha(i)$ that dominates the substituted component of α to N_2 (see Figure 4.19). It is possible for components above the substituted node to drift arbitrarily far up the d-tree and distribute themselves within domination edges, or above the root, in any way that is compatible with the domination relationships present in the substituted d-tree. This is the mechanism through which DTG handles long-distance dependencies.

Furthermore, there is a need to constrain the way in which the non-substituted components can be interspersed. This is done by either using appropriate feature constraints

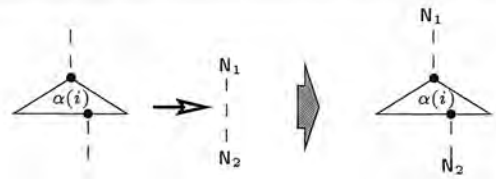


Figure 4.19: Insertion of a component $\alpha(i)$ in d-edge (N_1,N_2)

at nodes or by means of *subsertion-insertion constraints* which explicitly specify what components from what trees can appear within certain d-edges (we define them in Section 4.3.6). Another alternative for the subsertion-insertion constraints is to state them globaly for the whole grammar rather than locally for each d-edge in every elementary structure.

Figure 4.20 shows a linguistic example of subsertion:

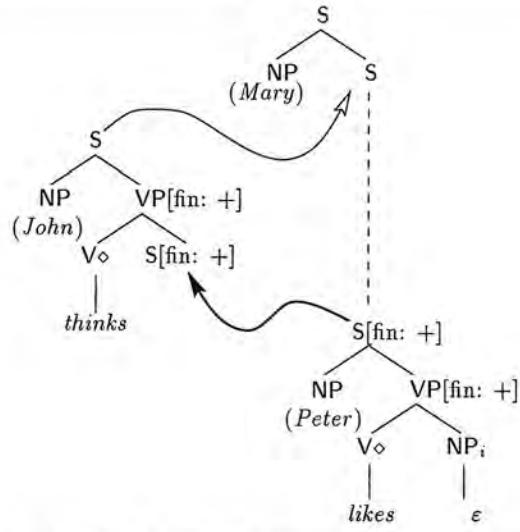


Figure 4.20: Linguistic example of subsertion

Subsertion as defined is a non-deterministic operation if the result of subsertion is to be a d-tree (an alternative is to have an encoding of the resulting alternatives). For example the two sentences:

1. *Hotdogs he claims Mary adores.* and

2. *He claims hotdogs Mary adores.*

have been constructed by subserting the d-tree of *hotdogs ... Mary adores* into the

d-tree of *he claims* at the same substitution node of the latter d-tree. We will take a closer look at this particular example in Section 4.3.4.

Subsertion can be viewed as a generalisation of the adjunction operation used in TAG. In fact subsertion can 'simulate' both adjuction and substitution. If we are subserting a tree β into a tree α and the substitution component is the top component (i.e., the the node from tree β that we equate with the substitution node in α is the root node of β) then there are no components that need to float above the substitution node in α and subsertion is reduced to substitution. Subsertion can be seen to simulate

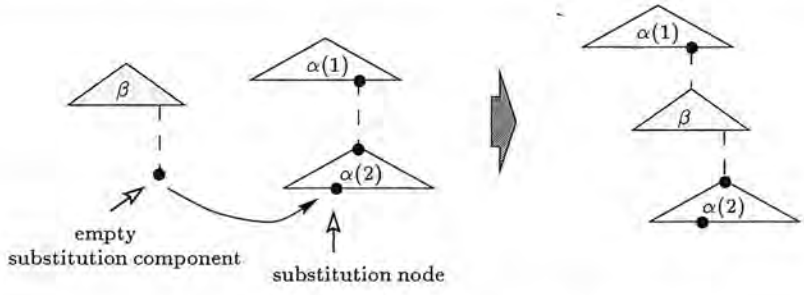


Figure 4.21: Subsertion and adjunction

adjunction at an abstract level (not in the mathematical sense of the operations being equivalent) in the sense that that splicing of a component internally into a tree can be achieved (see Figure 4.21).

4.3.3 Sister-adjunction

When a d-tree α is sister-adjoined at a node Y in a d-tree β the composed d-tree γ results from the addition to β of α as a new leftmost or rightmost sub-d-tree below Y. Note that sister-adjunction involves the addition of exactly one new immediate domination edge and that several sister-adjunctions can occur at the same node. Sister-adjoining constraints of a node (cf. Y) specify whether they will be right- or left-sister-adjoined.[11]

[11] The multiple adjunction defined in [Shieber & Schabes 94] can be seen as the precursor of sister-adjunction. They are strikingly close. Elaborating on the interactions between both operations is beyond the main focus of this thesis.

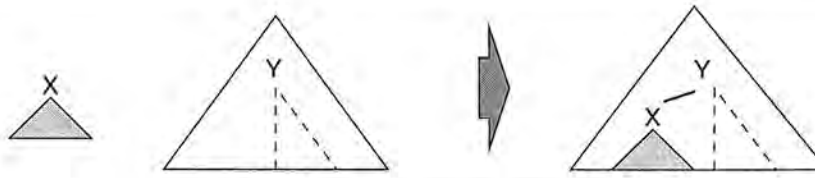


Figure 4.22: Sister-adjunction

Definition 4.1 (D-Tree Grammar)

A DTG *is a four tuple* $G = (V_N, V_T, S, D)$ *where* V_N *and* V_T *are the usual nonterminal and terminal alphabets,* $S \in V_N$ *is a distinguished nonterminal and* D *is a finite set of elementary d-trees.*

Definition 4.2 (Lexicalised DTG)

A DTG $G = (V_N, V_T, S, D)$ *is lexicalised iff each d-tree* $T \in D$ *has at least one terminal node.*

The elementary d-trees of a grammar G have two additional annotations: subsertion-insertion constraints and sister-adjoining constraints. These will be described below (Section 4.3.6) but let's first take a look at an example.

4.3.4 Example

In this section we consider how the following sentence:

Small, spicy hotdogs he claims Mary seems to adore

can be derived. This particular sentence is interesting because it involves: object extraction (topicalisation), sentential complementation and modification.

In Figure 4.23, we give a DTG that generates the sentence. Every d-tree is a projection from a lexical anchor. The label of the maximal projection is, we assume, determined by the morphology of the anchor. For example, if the anchor is a finite verb, it will project to S, indicating that an overt syntactic ("surface") subject is required for agreement with it (and perhaps case-assignment). Furthermore, a finite verb may optionally also project to S′ (as in the d-tree shown for *claims*), indicating that a wh-moved or topicalised element is required. The finite verb *seems* also projects to S, even though it does not itself provide a functional subject. In the case of the *to adore* tree,

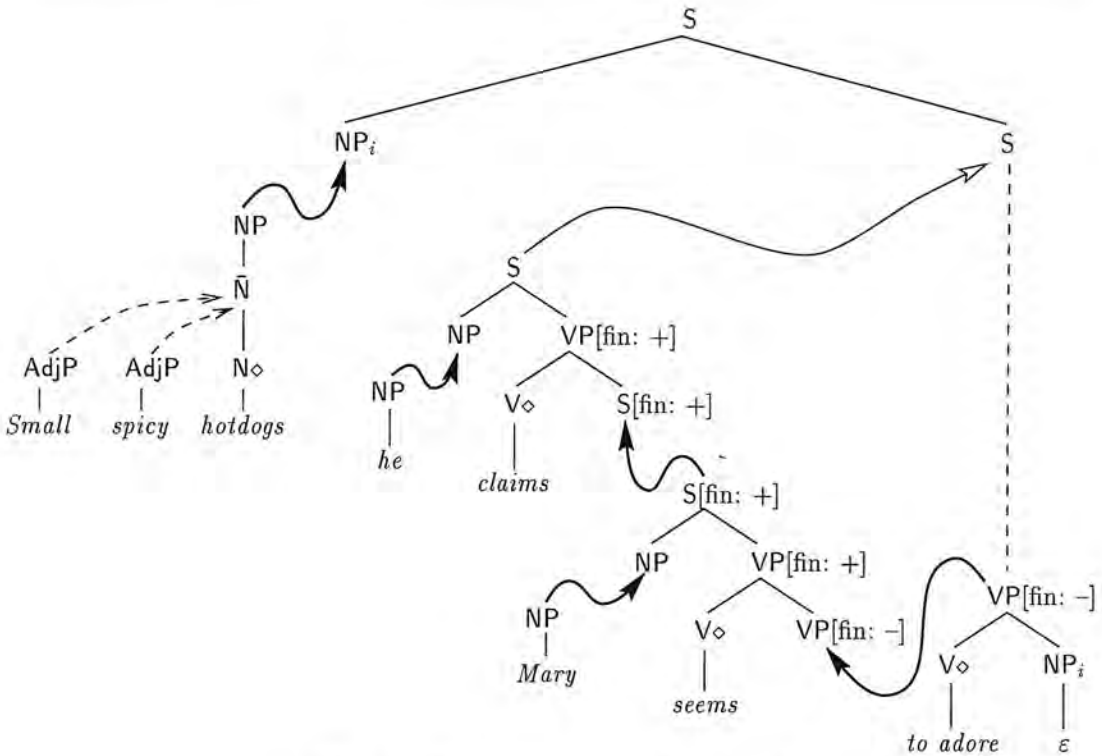


Figure 4.23: DTG grammar in operation

the situation is the inverse: the functional subject requires a finite verb to agree with, which is signaled by the fact that its component's root and frontier nodes are labelled S and VP, respectively, but the verb itself is not finite and therefore only projects to VP[-fin]. Therefore, the subject will have to raise out of its clause for agreement and case assignment. The direct object of *to adore* has wh-moved out of the projection of the verb (we include a trace for the sake of clarity).

We add subsertion-insertion constraints (SICs)[12] (which control what can go in d-edges) to ensure that the projections are respected by components of other d-trees that may be inserted during a derivation. A SIC is associated with the d-edge between VP and S node in the *seems* d-tree to ensure that no node labelled $\bar{S}$ can be inserted within it i.e., it can not be filled by a wh-moved element. In contrast, since both the subject and the object of *to adore* have been moved out of the projection of the verb,

[12] We postpone a discussion of subsertion-insertion constraints till Section 4.3.6. For the moment it suffices to mention that SICs are a mechanism through which DTG rules out the insertion of some components in certain d-edges.

the paths to these arguments do not carry any SIC at all.[13]

We now discuss a possible derivation. We start out with the most deeply embedded clause,[14] the *adore* clause. Before subserting its nominal arguments, we sister-adjoin the two adjectival trees (*small* and *spicy*) to the tree for hotdogs. This is handled by a SAC associated with the $\bar{N}$ node that allows all trees rooted in AdjP to be left sister-adjoined. We then subsert this NP structure (*Small, spicy hotdogs*) and the subject (*Mary*) at the appropriate nodes into the *to adore* d-tree. We subsert the resulting structure into the *seems* clause by substituting its maximal projection node, labelled VP[fin: -], at the VP[fin: -] frontier node of *seems*, and by inserting the component containing the subject (*Mary*) into the d-edge (S,VP[fin:+]) of the *seems* tree. Now,

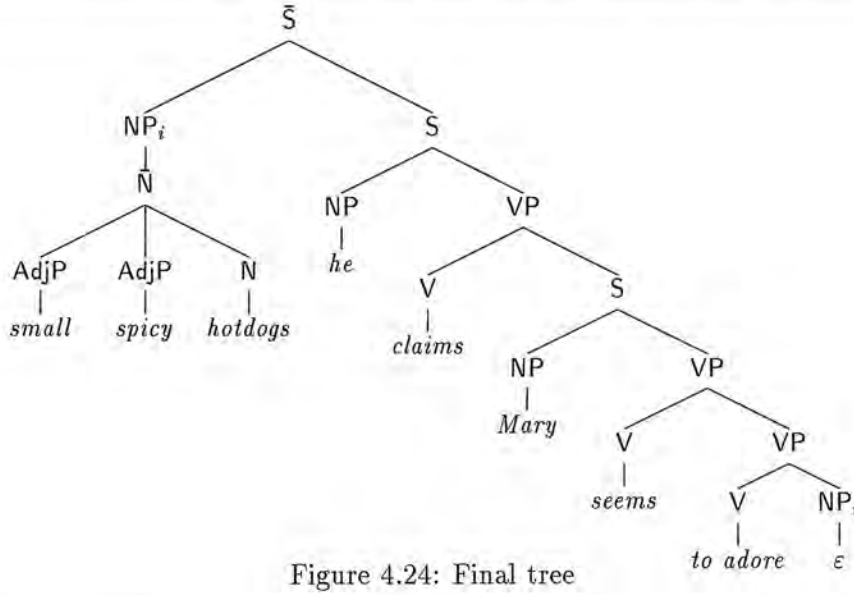


Figure 4.24: Final tree

only the S node of the *seems* tree (which is its maximal projection) is substitutable. Finally, we subsert this derived structure into the *claims* d-tree by substituting the S node of *seems* at the S complement node of *claims*, and by inserting the component containing the extracted object of *to adore* (which has not yet been used in the derivation) in the ($\bar{S}$,S) d-edge of the *claims* d-tree above its S node. The derived tree is shown in Figure 4.24.

[13] Island effects for wh-movement are enforced by using a [extract] feature on substitution nodes. This corresponds roughly to the analysis in TAG, where islandhood is (to a large extent) enforced by designating a particular node as the foot node [Kroch & Joshi, 1986].

[14] We are thus doing a bottom-up derivation. We consider bottom-up derivations in the context of generation in Section 5.10.2

Note that this is the only possible derivation involving these d-trees, modulo order of operations. To see this, consider the following putative alternate derivation. We first subsert the *to adore* d-tree into the *seems* tree as above, by substituting the anchor component at the substitution node of *seems*. We insert the subject component of *to adore* above the anchor component of *seems*. We then subsert this derived structure into the *claims* tree by substituting the root of the subject component of *to adore* at the S node of *claims* and by inserting the S node of the *seems* d-tree as well as the object component of the *to adore* d-tree in the ($\bar{S}$,S) d-edge of the *claims* d-tree. This

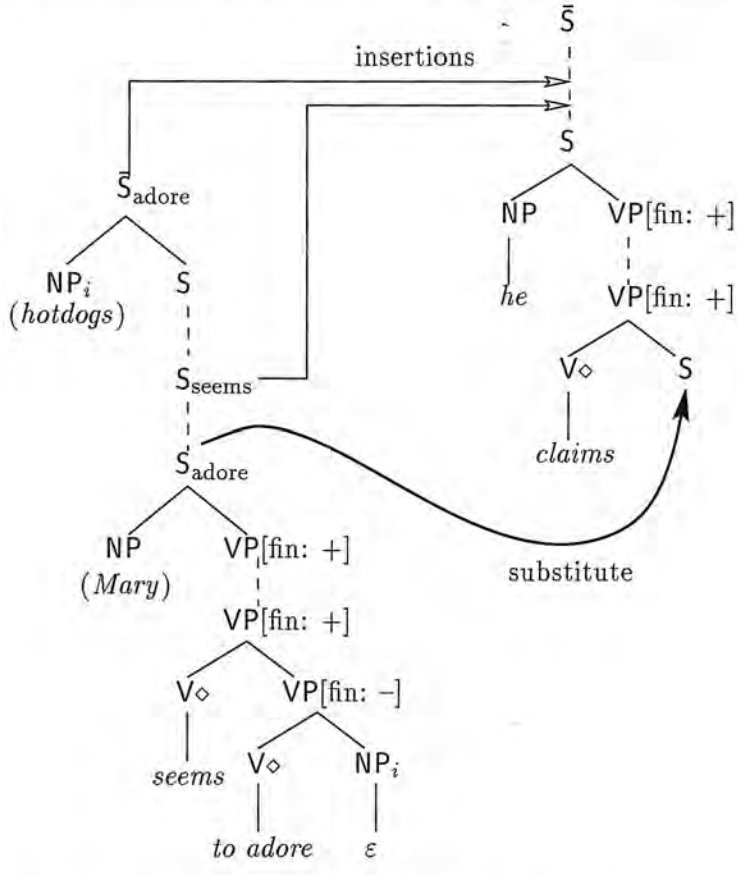


Figure 4.25: Incorrect derivation for the example

last operation is shown in Figure 4.25. The resulting phrase structure tree would be the same as in the previously discussed derivation, but the derivation structure is linguistically meaningless, since *to adore* would have been subserted into both *seems* and *claims*. However, this derivation is ruled out by the restriction that only substitutable components can be substituted: the subject component of the *adore* d-tree is not sub-

stitutable after subsertion into the *seems* d-tree, and therefore it cannot be substituted into the *claims* d-tree.

In the above discussion, substitutability played a central role in ruling out the derivation.[15] We observe in passing that the SIC associated to the d-edge in the *seems* d-tree also rules out this derivation. The derivation requires that the S node of *seems* be inserted into the ($\bar{S}$,S) of *claims*. However, we would have to stretch the edge over two components which are both ruled out by the SIC, since they violate the projection from the lexical item *seems* to its S node. Thus, the derivation is excluded by the independently motivated SICs, which enforce the notion of projection. This raises the possibility that, in grammars that express certain linguistic principles, substitutability is not needed for ruling out derivations of this nature.

4.3.5 Derivation structures

We first define subsertion-adjoining trees (SA-trees), which are partial derivation structures that can be interpreted as representing dependency information (Section 4.3.5.1). Then we augment them to get derivation graphs (Section 4.3.5.2).

4.3.5.1 Subsertion-adjoining tree

Consider a DTG $G = (V_N, V_T, S, D)$. In defining SA-trees, we assume some naming convention for the elementary d-trees in D (formally we will refer to them as α_1, α_2, etc.) and some consistent ordering on the components and nodes of elementary d-trees in D. If we think of components as big nodes then the structure that connects components (using d-links) is a tree.[16] Thus, we can identify components using tree addresses (in the tree of components) and we can refer to nodes within a component again using tree addresses for this component only.

For each i, we define the set of d-trees $T_i(G)$ whose derivations are captured by SA-trees of height i or less. Let $T_0(G)$ be the set D of elementary d-trees of G.

$$T_0(G) = \{t \mid t \in G\}$$

[15] We define substitutability in the next section (4.3.5).

[16] In actual linguistic examples the components form a chain (list).

Mark all of the components of each d-tree in $T_0(G)$ as being substitutable. Only components marked as substitutable can be substituted in a subsertion operation. The SA-tree for $\alpha \in T_0(G)$ consists of a single node labelled by the elementary d-tree name for α.

$$T_i(G) = T_{i-1}(G) \cup \Gamma_i$$

For $i > 0$ let $T_i(G)$ be the union of the set $T_{i-1}(G)$ with the set (Γ_i) of all d-trees that can be produced as follows. Let $\alpha \in D$ and let γ be the result of subserting or sister-adjoining the d-trees $\gamma_1, ..., \gamma_k$ into α where $\gamma_1, ..., \gamma_k$ are all in $T_{i-1}(G)$, with the subsertions taking place at different substitution nodes in α as the foot node. Only substitutable components of $\gamma_1, ..., \gamma_k$ can be substituted in these subsertions. Only the new components of γ that came from α are marked as substitutable in γ. Let $\tau_1, ..., \tau_k$ be the SA-trees for $\gamma_1, ..., \gamma_k$ respectively. The SA-tree τ for γ has a root labelled by the name for α and k subtrees $\tau_1, ..., \tau_k$. The edge from the root of τ to the root of the subtree τ_i is labelled by $l_i (1 < i < k)$ defined as follows. Suppose that γ_i was subserted into α and the root of τ_i is labelled by the name of some $\alpha' \in D$. Only components of α' will have been marked as substitutable in γ_i. Thus, in this subsertion some component $\alpha'(j)$ will have been substituted at a node in α with address η. In this case, the label l_i is the pair (j, η). Alternatively, γ_i will have been d-sister-adjoined at some node with address η in α, in which case l_i will be the pair (d, η) where $d \in \{\text{left,right}\}$.

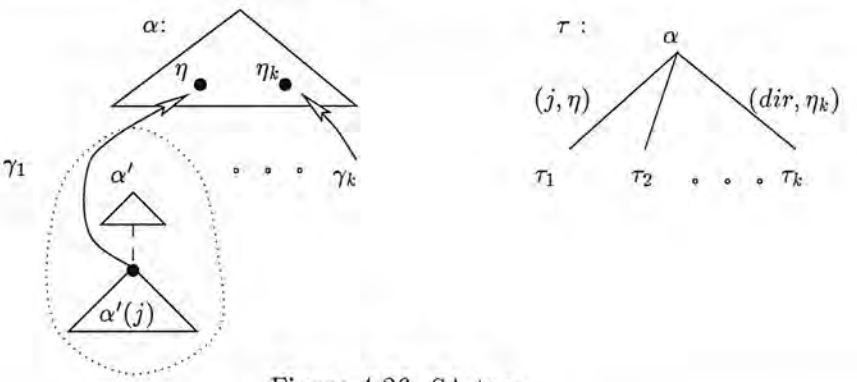


Figure 4.26: SA-tree

Figure 4.26 gives an SA-tree. The tree γ_1 has its component $\alpha'(j)$ substituted at node η in the tree α; the tree γ_k has been sister adjoined at node η_k in α.

Figure 4.27 shows an example SA-tree for the derivation in Figure 4.23 of the sentence in Figure 4.24.

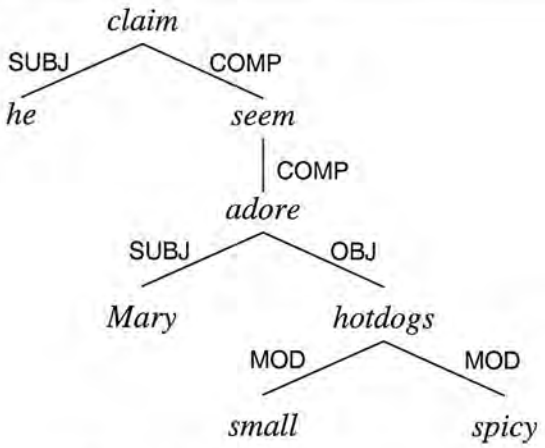


Figure 4.27: Example SA-tree

Given a d-tree β it might be possible to convert it into a tree by merging the nodes at the top and bottom of all d-links. Such merging can be done if the labels of the top and bottom are labelled by the same symbol. We call this process 'closing off' a d-tree and denote the result of this (conditional) function as $close\_off(\beta)$.

Definition 4.3 (DTG tree set)

The tree set $T(G)$ of all trees generated by G is:

$$T(G) = \{\gamma \mid \exists \gamma'.\exists i \geq 0 : \gamma' \in T_i(G) \;\&\; root(\gamma') = S \;\&\; yield(\gamma') \in V_T^* \;\&\; close\_off(\gamma') = \gamma\}$$

Definition 4.4 (DTG string language)

The string language $L(G)$ associated with a DTG G is the set of terminal strings appearing on the frontier of trees in the tree set $T(G)$.

4.3.5.2 Derivation graph

So far we have described SA-trees since they play such an important role in the motivation for introducing DTGs. We now describe a structure that can be used to encode a DTG derivation. A derivation graph for $\gamma \in T(G)$ results from the addition of insertion edges to a SA-tree for τ for γ. The location in γ of an inserted elementary component $\alpha'(i)$ can be unambiguously determined by identifying the source of the node (say the node with address η_2 in the elementary d-tree α) which the root of this occurrence of $\alpha'(i)$ is merged with when d-edges are removed. The insertion edge will relate the two

(not necessarily distinct) nodes corresponding to appropriate occurrences of α' and α and will be labelled by the pair (i, η_2) (see Figure 4.28).

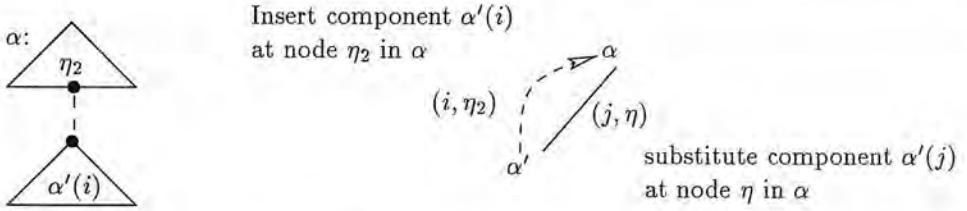


Figure 4.28: DTG derivation graph

4.3.6 Constraining subsertion and sister-adjunction

Each d-edge in elementary d-trees has an associated *subsertion-insertion constraint* (SIC). A SIC is a finite set of elementary node addresses (ENAs). An ENA specifies some elementary d-tree $\alpha \in D$, a component of α and the address of a node within that component of α. If a ENA η is in the SIC associated with a d-edge between $\eta 1$ and $\eta 2$ in an elementary d-tree α then η cannot appear properly within the path that appears from $\eta 1$ to $\eta 2$ in the derived tree $\gamma \in T(G)$.

Each node of elementary d-trees has an associated *sister-adjunction constraint* (SAC). A SAC is a finite set of pairs, each pair identifying a direction (left or right) and an elementary d-tree. A SAC gives a complete specification of what can be sister-adjoined at a node. If a node (with tree address) η is associated with a SAC containing a pair (d, α) then the d-tree α can be d-sister-adjoined at η. By definition of sister-adjunction, all substitution nodes and all nodes at the top of d-edges can be assumed to have SACs that are the empty-set. This prevents sister-adjunction at these nodes.[17]

We are now in a stronger position to appreciate the additional advantages of DTGs over TAGs for generation. The SA-trees reflect the semantic structure and wherever on the semanic level there are complements in the syntax subsertion is used. Semantic modifiers are handle by sister adjunction. This uniformity is possible because of the flexibility of the syntax. Generation in DTGs is simple because relating the semantics to an SA-tree is trivial. This is more of a problem for TAGs. Consider the corresponding

[17] Although in the final derived tree these nodes will be merged with nodes that can be sister-adjoined at (i.e., modified).

TAG derivation to the DTG one in Figure 4.27 which is given in Figure 4.23. Producing a derivation from the input is the route that FLAUBERT [Danlos & Meunier 96] and VMGeCo [Becker *et al.* 98] follow. Alternatively, when considering a complement in a TAG-based generator a check has to be made whether the complement corresponds to an auxiliary tree in which case it is the matrix clause that is adjoined into the embedded clause. This is how PROTECTOR-95 [Nicolov *et al.* 95] operates.

4.3.7 Formal power of DTGs

In our linguistic analyses we have used a lot of the TAG analyses. Yet, surprisingly as it may be TAGs and DTGs are not formally equivalent [Rambow *et al.* 96]. Both TAG and DTG have constrained generative power that exceeds that of CFG, yet there are languages that one can generate but not the other and vice versa. DTG can generate Mix_k for any k.[18] Mix_k for $k \leq 3$ cannot be generated by TAG. DTG can "count" to any natural number k. That is languages $L_k = \{a_1^n, \ldots a_k^n\}$ for any k can be generated DTG. TAG however can only count up to three.[19] On the other hand Rambow, Vijay-Shanker and Weir [Rambow *et al.* 96] conjecture that DTG cannot generate the copy language[20] which TAG can. More details on this can be found in [Rambow *et al.* 96].

4.3.8 Descriptions of d-trees and subsertion-insertion algorithm

In this section we describe the particular algorithm that we use to perform the subsertion operation. The algorithm assumes a particular representation of (description of) trees. A description of a tree is a set of the dominance relations between the nodes in the d-tree.

For example the d-tree (description) in Figure 4.29 is represented as four (difference) lists of (1) immediate dominance, (2) dominance, (3) linear precedence constraints between the nodes in the tree and (4) labelling of the nodes (see Figure 4.30).

---

[18] Mix_k is the set of strings with equal number of occurrences of each $a_i (1 \leq i \leq k)$ appearing in any order.

[19] We do have an implementation of DTG that generates Mix_k but the details of this go beyond the central point of this thesis.

[20] $\{ww \mid w \in \{a, b\}^*\}$.

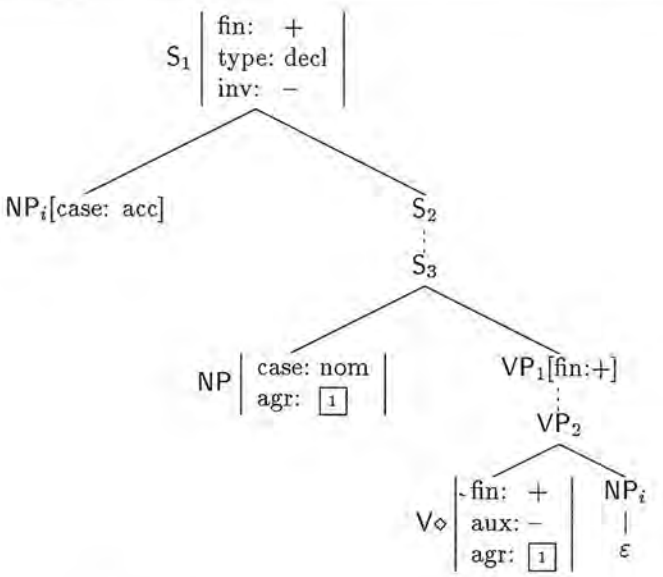


Figure 4.29: Extracted object construction

where `Idx` is a unique index of the (copy of the) tree, every node is associated with a node name (e.g., `np(Idx)`) which involves the index of the tree `Idx` so that nodes from different trees (or copies of the same tree) are not confused. The feature labels list contains pairs of node name and (term) encoding of the feature structure associated with the node.

Current TAG implementations use trees as a data structure to represent the elementary trees as well as the results of adjunction (and substitution). In the case of adjunction the result has to be reconstructed. Our use of descriptions simplifies the algorithms for combining two d-trees. In addition we can work entirely with d-tree descriptions and not resolve the actual d-trees till the very end (a d-tree description can correspond to a number of trees). Thus we can get more paraphrasing power doing less computations. In our actual implementation we work with minimal descriptions. We do not have a dominance relationship if it can be derived from other relationships.[21] The use of descriptions is consistent with new reformulations of DTG which use a tree logic to describe sets of trees. Such mechanisms have been considered elsewhere [Backofen 94]

[21] It is interesting to consider alternative data structures (encodings of d-trees) that might allow the implementation of the subsertion-insertion in constant time no matter what the size of the input structures. Such an operation has to be non-deterministic because of the nature of the composition operations. In terms of the strings we can think of the elementary d-trees as strings with holes in them that can be further specified.

```
Idx:
[ s1(Idx)  +> npi(Idx),          % Immediate dominance constraints
  s1(Idx)  +> s2(Idx),
  s3(Idx)  +> np(Idx),
  s3(Idx)  +> vp1(Idx),
  vp2(Idx) +> v(Idx)    |X]-X :
[ s2(Idx)  ++> s3(Idx),          % dominance constraints
  vp1(Idx) ++> vp2(Idx) |Y]-Y :
[ npi(Idx) < s2(Idx),            % LP constraints
  np(Idx)  < vp1(Idx)   |Z]-Z :
[ s1(Idx) : s(fin,decl,noninv),  % feature
  s2(Idx) : s(_,_,_),            %   labelling
  s3(Idx) : s(_,_,_),
  npi(Idx): np(_,_,acc),
  np(Idx) : np(Num,Per,nom),
  vp1(Idx): vp(_,fin),
  vp2(Idx): vp(_,_),
  v(Idx)  : v(Num,Per,fin,nonaux) |W]-W
```

Figure 4.30: Representations of tree descriptions

and we will not describe them here.

procedure *subsert*(*TreeA*, *NodeA*, *TreeB*, *NodeB*, *Resulting_Tree*);
Input Args:

TreeA:	Tree that is subserted (list of domination constraints)
NodeA:	Node that is be subserted
TreeB:	Tree that is subserted into (list of domination constraints)
NodeB:	Substitution node

Output Args:

Resulting_Tree:	A tree conforming to the union of the domination constraints of TreeA and TreeB after nodes NodeA and NodeB have been identified

```
begin
unify NodeA and NodeB;
if      NodeA is the root of TreeA
then        Resulting_Tree :=
            union of the domination constraints of TreeA and TreeB;
else
    begin
        LinkA := domination link dom(NA, NodeA) in TreeA;
        Root := root of the component in which NodeB is in TreeB;
        replace LinkA in TreeA with the link D1 which is dom(NA, Root);
        if Root is not the root of the whole tree TreeB
              i.e., there is a link D2 dom(NB,Root) in TreeB
        then
            begin
                Domination_Links := [D1,D2];
                Semi_Tree := union of the domination constraints
                      of TreeA and TreeB;
                resolve(Root,Domination_Links,Semi_Tree,Resulting_Tree);
            end;
        else
            Resulting_Tree := union of the domination constraints
                  of TreeA and TreeB;
    end;
end;
```

procedure *resolve*(*Node*, *Domination_Links*, *Semi_Tree*, *Resulting_Tree*);
Input Args:

Node:	*Node* is the root of a component
Domination_Links:	list of two domination links of the kind: $dom(X, Comp\_Root)$ *Comp_Root* is the root of the component from which we are starting
Semi_Tree:	list of domination constraints

Output Args:

Resulting_Tree:	A tree conforming to the domination constraints in Semi_Tree

begin
order the links in *Domination_Links*;
choose one link (in the order of specification)
Chosen := the chosen d-link — $dom(N, R)$;
Other := the other d-link;
New_Root := the root of the component of the node N;
if
New_Root is dominated by another node
i.e., there is a link $L1$ $dom(\_, New\_Root)$ in *Semi_Tree*
then
replace link *Other* : $dom(Node, R)$ with
$L2 : dom(Node, New\_Root)$ in *Semi_Tree*;
$New\_d\_links := [L2, L1]$;
resolve(*New_R*,*New_d_links*,*Semi_Tree*,*Resulting_Tree*)
else
replace link *Other* : $dom(Node, R)$ with
$dom(Node, New\_Root)$ in *Semi_Tree*;
Resulting_Tree := *Semi_Tree*;
end;

The nodes in the syntactic structure are feature structures and we use unification to combine two syntactic nodes [Kay 83].

4.4 Conclusions

In this chapter we talked about syntactic representations and considered formalisms that exibit more power than CFG. We looked closely at Tree-Adjoing Grammars and considered variants of these (quasi-trees) that lead us to D-Tree Grammars. In fact we have presented the most complete definition of DTG. Although TAG and DTG are very close they are not equivalent. Historically we started with an implementation

that used TAG and later developed a version using DTG. We were able to reuse to a large extent the linguistic analyses. This was the first implementation of DTG and the first use of DTG for generation. Indeed our implementation is currently the only one.[22] The generation algorithm for DTG is simpler due to the more uniform treatment of the composition operations on the syntactic and semantic side (every time on the semantic side we have complementation we use subsertion on the syntactic representation and wherever we have modification on the semantic side we use sister adjunction at the syntax level). DTG can be lexicalised which means that once lexical items are found we need not worry about any other grammatical structures and just concentrate on how we can put the d-trees together in a well-formed derivation. This has important ramifications for parsing and generation.

SUMMARY

- D-Tree Grammars are based on a long tradition of TAG.
- DTGs have a larger domain of locality. They localise *complement-of* and *filler-gap* dependences.
- DTGs can be strongly lexicalised.
- The syntactic operations in DTGs mirror more closely the semantic operations of complementation and modification.
- DTGs can be seen to compile structures which in other frameworks need to be derived at run time.
- The generation algorithm for DTGs is simpler than that for TAGs.

In the next chapter we explain how semantic annotations can be added to the elementary trees and how DTG is used in generation.

[22] In a large project on Robust Parsing of English with DTG we have reused a lot of the modules of our generator.

Chapter 5

Declarative Generation from Non-Hierarchical Structures

This chapter and the next one describe the main contributions of this work. We describe a generation technique embodied in a generation system called PROTECTOR (approximate **PRO**duction of **TE**xts from **C**onceptual graphs in a declara**T**ive Frame-w**OR**k). Our generation methodology:

1. does not assume a hierarchical nature of the input and thus allows us to look at a more general version of the sentence generation problem where one is not pre-committed to a choice of the syntactically prominent elements in the initial semantics.

2. uses the notion of approximate matching of semantic structures. In order to constrain this we employ 'upper' and 'lower' bounds on the semantic input which express the least and the most that should be conveyed. This constrains in a principled way the additions to as well as the omissions from the input semantics the generator can make.

The assumptions that we make and the facilities that we have developed within the PROTECTOR generation system directly address the issue of increasing the paraphrasing power of the generator. As we have argued in Chapters 1 and 2 increased paraphrasing power means that the input is more 'semantic' and that the generator could use additional constraints to select more appropriate paraphrases in different situations. All this is particularly relevant for multilingual generation, in the context of MT, and

in cases where generators are given input from non-linguistic applications.

In addition PROTECTOR can be characterised along the following dimensions:

1. We develop a *declarative* framework for generation. The relation between meaning and form is stated in a declarative manner, i.e., this specification does not depend on a particular way it is going to be used but expresses how a particular semantic pattern can be rendered using a certain syntactic construction. Declarative grammars[1] allow the use of a grammar which was initially developed for parsing to be used for generation too (such grammars are called bidirectional). From a generation perspective a declarative grammar can be used using different techniques, i.e., using different generation strategies. In this chapter we describe our declarative notion of derivation. Different possible generation strategies are possible within our framework and we concentrate on one particular regime of processing—top-down generation (we also provide an abstract specification for bottom-up generation). The increased paraphrasing power also means that the search space is now larger and we introduce a number of mechanisms for controlling the search.

2. Unlike most of the existing generation systems which perform surface realisation we consider lexical choice as an integral part of the generation process. Different ways of grouping/packaging concepts together in a lexical item provides one of the important mechanisms for paraphrasing. Performing lexical choice within the generation stage allows for intricate interactions between lexical and syntactic decisions to be considered.

3. Our generation system performs a 'delayed' inflection of lexical items. After the abstract lexeme is chosen its particular inflected form is generated only after all required syntactic information becomes available. An exception to this rule is the case where there is only one form known to the generator. This is a form of deterministic goal expansion.

[1] Here by grammar we mean the broader notion of a specification of how syntax and semantics are related.

This chapter presents the general declarative model for generation and describes one technique for sentence generation (we show a top-down strategy). The semantic input to the generator is cast as a conceptual graph and the syntactic structures that the system manipulates are d-trees. We'll assume familiarity on behalf of the reader with Conceptual Graphs (introduced in Chapter 3) and D-Tree Grammars (introduced in Chapter 4).

Overview

We proceed with describing the knowledge sources available to the generator (Section 5.1): conceptual ontology (5.1.1), input semantics (5.1.2), 'lower' and 'upper' semantic constraints (5.1.3), input syntactic and correspondence constraints (5.1.4) and mapping rules that relate the semantics to the syntax (5.1.5). Next in Section 5.2 we discuss the outputs of the generator: the final syntactic structure (5.2.1) and the corresponding semantics (5.2.2). This is followed by a presentation of a top-down generation algorithm (Section 5.3) which includes pseudo-code for the top-down strategy (5.3.4). We show a step-by-step illustration of the generation of one sentence (Section 5.4). Further semantic aspects of the generation are discussed in (Section 5.5). We discuss lexical choice in our model (5.6), and the ordering of modifier trees (5.7). We then show an example of paraphrases using approximate matching in Section 5.8 and talk about generation of follow-up sentences (5.9). We provide a declarative specification of our notion of derivation in Section 5.10. We give an abstract specification of the top-down derivation (5.10.1), bottom-up derivation (5.10.2) and consider the more sophisticated notion of matching semantic structures (5.10.3). We then talk about our implementation (5.11) and the coverage of our grammar (5.12). We discuss some issues related to the proposed technique (Section 5.13) and give conclusions in (Section 5.14).

5.1 Knowledge sources

This section presents the different types of knowledge the generator uses in order to produce a sentence. Figure 5.1 shows the overall generation architecture we assume.

The main input to the generator is what we call the *Input Semantics (InputSem).*

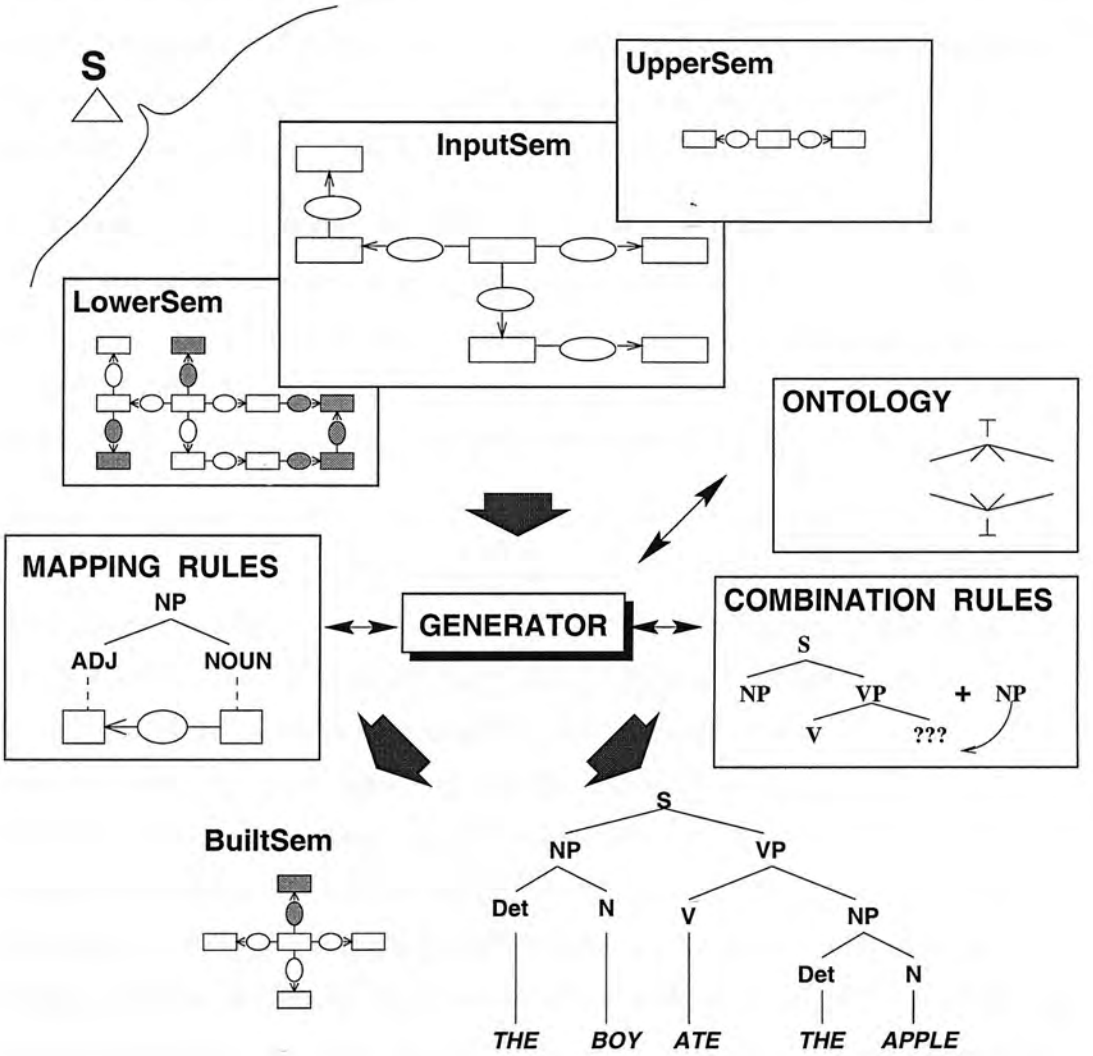


Figure 5.1: General view of the architecture

This is the message. There are two additional semantic structures which act as additional constraints and whose existence is motivated by the fact that PROTECTOR uses approximate matching of semantic structures. These structures are the *Lower Semantics (LowerSem)* and *Upper Semantics (UpperSem)*. They constrain how overgeneral and how overspecific the generator can be. Another input the generator considers is a structure that constrains the kind of syntactic output that has to be produced. Thus, for instance, the generator can produce a sentence or a nominal group from the same input. In Figure 5.1 we have expressed the desired syntactic category as a tree-description with root S and an empty tree underneath it: 

The output of the generator is a syntactic tree and a semantic structure corresponding to the generated sentence. One of the central points that this thesis makes is that generators can be forced to say more or less than the input semantics requires, and producing the semantic structure corresponding to the generated sentence is vital in order for the generator to keep track of its over-commitments and underspecificity.

During the generation process the system uses knowledge about how pieces of semantic structure are related to syntactic constructs. We call this knowledge *mapping rules*. The generator needs to match the semantic patterns that it knows about, in the form of the semantic parts of mapping rules, against particular configurations in its input. In this process of matching knowledge about which concepts are compatible is essential. In our system this is represented as the conceptual ontology. The generator also has to know how to combine syntactic structures and what are the semantic ramifications concerning the semantic structure corresponding to the combined syntactic structure. In the figure we have labeled this knowledge source as *combination rules*. When combining structures the generator uses knowledge about which semantic concepts correspond to the same concept in the input semantics. On the syntactic side the structures have to be licensed by the syntactic grammar.

5.1.1 Conceptual ontology

The conceptual ontology consists of the hierarchy of concepts and the hierarchy of (conceptual) relations. Both are multiple inheritance hierarchies. The concept hierarchy is similar to a part of the Upper Model and Domain Model of PENMAN [Bateman *et al.* 89]

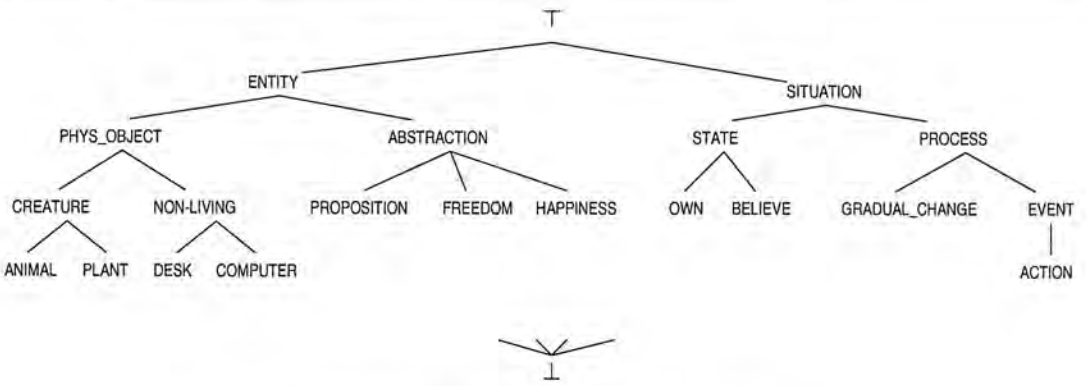


Figure 5.2: Conceptual ontology

or the Upper Cyc Ontology [Cyc96].

The conceptual ontology is not part of the input that is fed into the generator but is used when two semantic structures (typically a configuration from the input and a semantic part of a mapping/grammar rule) are matched/combined (or checked for compatibility). Corresponding concepts in the two semantic structures must be joinable and the generator can determine this by looking at the conceptual hierarchy to see whether the types of the two concepts have a common subtype.

There is also a separate hierarchy for conceptual relations. We make use of conceptual relations like the ones in Table 5.1.

ROLE	ABREV.	EXAMPLE
AGENT	AGNT	*John built a house*
BENEFICIARY	BEN	*John bought Mary flowers*
EXPERIENCER/GOAL	EXPR/GOAL	*Romeo loves Juliet*
INSTRUMENT	INSTR	*Alexander cut through the knot with his sword*
PATIENT/THEME	PTNT/THEME	*He bought a computer*
LOCATIVE	LOC	*He lives in Edinburgh*
...	...	

Table 5.1: Conceptual relations

The choice of concepts and relations is important in as much as our generation system has to use a particular ontology in order to describe the semantic patterns and how they correspond to syntactic structures. Yet, the choice of a particular ontology over another is an issue we are not concerned with here. The generation mechanisms that

we develop are general and do not depend on this choice.

5.1.2 Input semantics

The input semantics is a conceptual graph representing what should be expressed. Figure 5.3 gives an example of what input structures look like. The CG in the figure describes a situation which can be expressed as:

> *John bought Mary a £150 ticket from Edinburgh to Stuttgart from a friendly man with a mustache working at a student travel centre.*

There are, of course, many other ways in which this semantics can be expressed.

It is worth pointing out that in this thesis we are not looking at the distinctions between given and new information in the input semantics—we assume all information to be new (as in cases of describing new situations, etc.). There are good proposals in the NLG literature for generating definite descriptions, referring expressions, anaphors and the available techniques need to be integrated in PROTECTOR in the future.

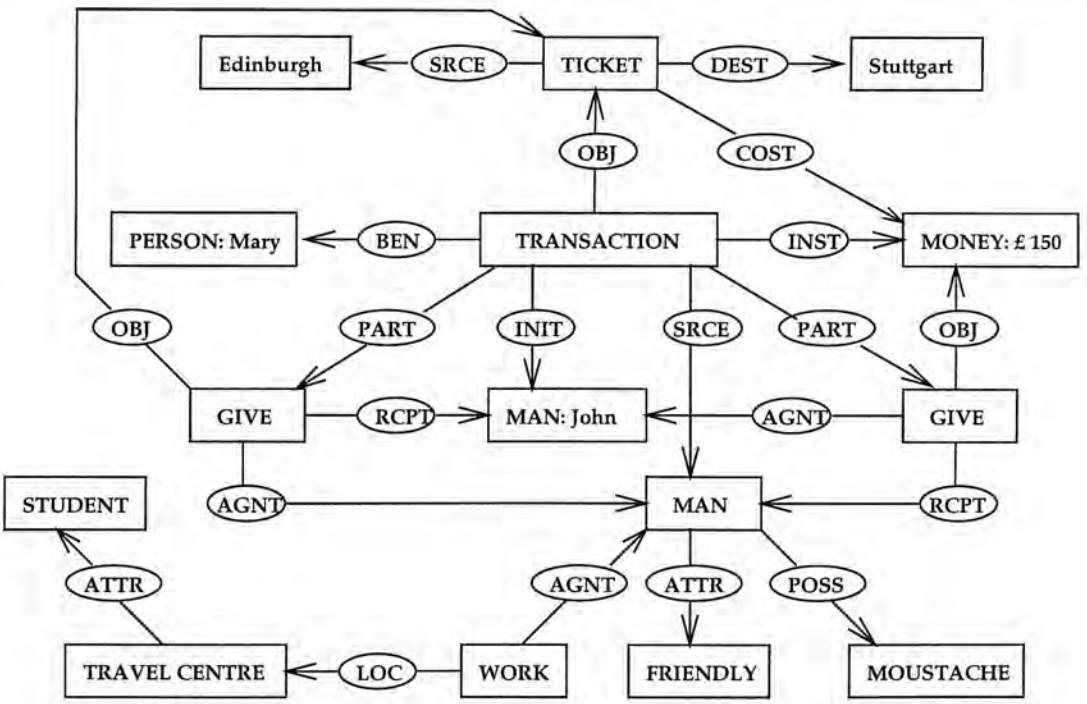


Figure 5.3: Example of a (complex) input semantics

5.1.3 Semantic constraints: Lower and Upper semantics

The generator assumes it is given as input an input semantics (*InputSem*) and 'boundary' constraints for the semantics of the generated sentence (*BuiltSem* which in general is different from *InputSem*[2]). The boundary constraints are two graphs (*UpperSem* and *LowerSem*) which convey the notion of the least and the most that should be expressed. We assume the following condition holds:

$$LowerSem \leq InputSem \leq UpperSem^3 \tag{5.1}$$

Needless to say it would be contrary to our intuition if things were otherwise. Figure 5.4 illustrates the input semantics and its upper and lower constraints (we use a diagrammatic notation to describe the semantics which is actually encoded using conceptual graphs; pictorially we also use the metaphor of embeddedness to represent specialisation).

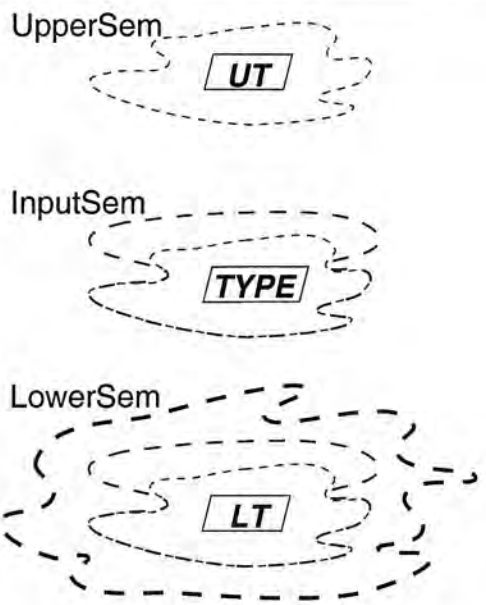


Figure 5.4: Boundary semantics

Every concept in *UpperSem* has a corresponding concept in *InputSem* and every con-

[2] This can come about from a mismatch between the input and the semantic structures expressible by the generator.

[3] The notation $G_1 \leq G_2$ means that G_1 is subsumed by G_2 (i.e., G_1 is more specific than G_2).

cept in $InputSem$ has a corresponding concept in $LowerSem$.[4] These correspondences can be thought as given with the input to the generator. We define them precisely later in Section 5.10.3.1.

We want $BuiltSem$ to be a specialisation of $UpperSem$ and a generalisation of $LowerSem$ (in terms of the conceptual graphs that represent them):

$$LowerSem \leq BuiltSem \leq UpperSem \tag{5.2}$$

The generator uses $InputSem$ as the main structure that guides the generation process. However, if the system needs to deviate from $InputSem$, then such deviations need to be checked that they respect condition 5.2 at the end of the processing. There are two ways in which the generator can 'introduce' (locally) additional semantics:

1. specialising a concept: it can express a concept using an expression whose semantics is a more specific concept; or
2. adding new concepts: a language specific construction conveys more concepts than are available in $InputSem$.

Similarly the generator can be making (local) generalisations by:

1. generalising a concept; or
2. missing out chunks from $InputSem$.

The reason why we have been describing the specialisations and generalisations locally is because in general $InputSem$ and $BuiltSem$ might be incomparable (i.e., neither one subsuming the other). We do not want to impose both coherency ($InputSem \leq BuiltSem$) and completeness ($BuiltSem \leq InputSem$) which taken together would guarantee that $BuiltSem = InputSem$. This is because we want to allow cases of mismatches between both structures.

The generator is not free to make arbitrary generalisations/specialisations to $InputSem$ and the constraint 5.2 is one way to restrict what is possible. We now present additional constraints.

[4] This follows from the definition of subsumption given in Chapter 3.

5.1.3.1 The role of the input semantics

In this section we examine closely the valid specialisations and generalisations that the generator is allowed to make.

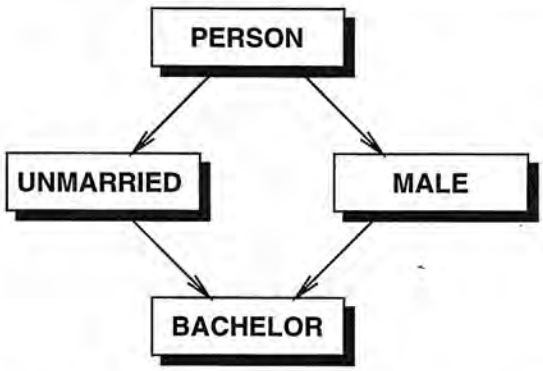


Figure 5.5: Example motivating additional constraints

Let us consider a type hierarchy such as that in Figure 5.5. If the input semantics contained the concept [UNMARRIED], and the concept in the upper semantic constraint was [PERSON], the concept in the lower semantic constraint [BACHELOR], then it would be awkward if the generator produced *male* (for illustrative purposes we assume trivial realisations for the concepts). A corollary of constraint 5.2 is that every concept $C_{BuiltSem}$ in the built semantics must satisfy:

$$C_{LowerSem} \leq C_{BuiltSem} \leq C_{UpperSem} \tag{5.3}$$

where $C_{LowerSem}$ and $C_{UpperSem}$ are corresponding concepts of $C_{BuiltSem}$ in the lower semantics and the upper semantics.[5] Alas, this constraint is too weak and it does not rule out the above case (because $BATCHELOR \leq MALE \leq PERSON$).

In order to rule out such siblings we would want the expression used by the generator to convey a concept which is in the generalisation or specialisation space of the original concept in the input semantics (see Figure 5.6).

More precisely this constraint can be expressed as:

[5] Again the reader is referred to Section 5.10.3.1. There we show how these correspondences are established.

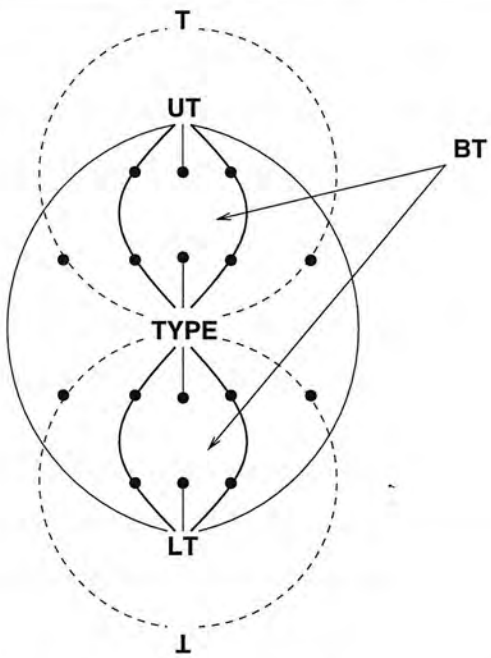


Figure 5.6: Constraints on the built type

$$\begin{array}{c} C_{LowerSem} \leq C_{BuiltSem} \leq C_{InputSem} \\ or \\ C_{InputSem} \leq C_{BuiltSem} \leq C_{UpperSem} \end{array} \tag{5.4}$$

Relating these definitions to Figure 5.5 we have: $C_{UpperSem} = \boxed{\text{PERSON}}$, $C_{LowerSem} = \boxed{\text{BACHELOR}}$ and $C_{InputSem} = \boxed{\text{UNMARRIED}}$. So $C_{BuiltSem} = \boxed{\text{MALE}}$ is not allowed.

However, because of constraint 5.1

$$LowerSem \leq InputSem \leq UpperSem$$

it might happen that there are concepts in $InputSem$ which do not have counterpart concepts in $UpperSem$. In this case we require that:

$$\begin{array}{c} C_{LowerSem} \leq C_{BuiltSem} \leq C_{InputSem} \\ or \\ C_{InputSem} \leq C_{BuiltSem} \leq \boxed{\text{T}} \end{array} \tag{5.5}$$

An additional complication is the case when the generator adds to $BuiltSem$ a concept which does not have a counterpart concept in the original $InputSem$ (i.e, we don't have $C_{InputSem}$). Obviously there wouldn't exist $C_{UpperSem}$ either. In this case we require that the generator stays more general than $LowerSem$:

$$C_{LowerSem} \leq C_{BuiltSem} \tag{5.6}$$

We delay a discussion of when exactly in the generation process such constraints are checked until we have introduced the notion of derivation (see Section 5.10.3.1).

The goal of the generator is to produce a sentence whose corresponding semantics is as close as possible to the input semantics, i.e., the realisation adds as little as possible extra material and misses as little as possible of the original input. In generation, similar constraints have been used in the generation of referring expressions where the expressions should not be too general so that discriminatory power is not lost and not too specific so that the referring expression is in a sense minimal [Reiter & Dale 92]. Our model is a generalisation of the paradigm for choosing nouns presented in [Reiter 91] where (i) the generator must "convey sufficient information to fulfill the system's underlying communicating goals (the *upperSem* in PROTECTOR); and (ii) the generator makes a distinction between what it "knows about the object being lexicalised and what it wishes to communicate about this object." We return to how $UpperSem$ and $LowerSem$ are actually used in Sections 5.10.3 and 5.5.

5.1.4 Input syntactic and correspondence constraints

Standardly sentence generators take a clause sized semantic input and generate a sentence from it. Often, however, there are other phrases with top level categories different from a sentence that are desired as output. Given that sentence generators in their internal workings do produce phrases of other types too there is nothing that prevents a sentence generator from generating non-sentential phrases from its input. PROTECTOR also expects a constraint on the type of syntactic structure that it is has to produce (we have marked these constraints as '**S**' in Figure 5.1). This often contains a single syntactic node which is required to be compatible with the root of the syntactic tree that is produced. More elaborate constraints can force certain correspondences be-

tween syntactic nodes and elements from the semantics, or refer to the structure of the final syntactic tree. Later on we refer to this mixed semantic-syntax input structure as *Partial*. Initially *Partial* can be seen as a starting description of what the final structure to be produced by the generator is to be like. Obviously whatever the generator does it should not violate the constraints initially provided by *Partial*. *Partial* can only be extended monotonically. After the generator reaches a point where the current structure is entirely subsumed by the initial syntactic and correspondence constraints, checks for violations no longer need to be performed because of the monotonic character of the extensions.

5.1.5 Mapping rules

Every generator must have a way of relating semantic information to the linguistic means of expressing it. While some generation paradigms obscure the *semantics* $\leftrightarrow$ *syntax* relation, we want to have an explicit representation for it stated in a declarative fashion.

Mapping rules (MRs) state how the semantics is related to the syntactic representation, i.e., they specify the *semantics* $\leftrightarrow$ *syntax* relation. We do not impose any intrinsic directionality on the mapping rules and view them as declarative statements. Although we will not use mapping rules in the direction of *syntax* $\rightarrow$ *semantics* (i.e., for understanding) we think of them as being free from control information so that mapping rules can be used with different generation strategies. In order to cut down on the number of mapping rules we impose the condition of minimality of the description of mapping rules—only those details from both semantic and syntactic structures that are relevant for the mapping are stated. That implies that the mapping rules will link partial semantic structures with partial syntactic structures.[6] The minimality condition for MRs makes the semantic domains even more language dependent—the semantic domains reflect the way the natural language carves up the input semantics (if one assumes a top-down view).

[6] Our imposing of such a minimality condition is similar in spirit to the *economy of expression* principle in LFG.

5.1.5.1 Types of mapping rules

We distinguish between two types of MRs:

1. **lexical:** directly map semantics to a syntactic structure (d-tree description) representing a complete surface form/phrase extensions (all leaf nodes are terminal). Used for lexical items (words), fixed idioms and canned text.

2. **non-lexical:** map a piece of semantics to a syntactic structure which is a d-tree description and some of the frontier nodes of d-this tree (at least one) are marked with a non-terminal. Most verbs anchor non-lexical mapping rules; the frontier non-terminals of these constructions are the subject and objects of the verb.

The non-lexical MRs can be roughly seen as the parallel to classical grammar rules while the lexical ones as the counterpart to a specification of the lexicon (terminal symbols or unit clauses in logic programming). The analogy doesn't go much further than this because in lexicalized DTGs some lexical items (those with complex subcategorisation information, e.g., verbs) only exist in non-lexical mapping rules.

5.1.5.2 Abstract representation of mapping rules

A mapping rule is a data structure which contains the following information:

name: represents the name of the MR;

applicability semantics: this is a conceptual graph which states the semantic information 'covered' by this construction. It may have annotations about the *head* of the graph and which nodes *obligatorily* have to be in the maximal projection when such a graph is maximally joined (to the goal semantics);[7]

d-tree: this contains the syntactic information (i.e., how the applicability semantics can be expressed);

[7] For readability's sake we delay a discussion of what exactly the head and obligatory annotations mean until the next section when we consider some of the extensions to the model.

linking relation: a relation (in the mathematical sense) between nodes in the d-tree and concepts in the applicability semantics; In the direction *d-tree nodes* → *applicability semantics nodes* the linking relation is a partial function.[8]

generation goals: this a (possibly empty) list of all the frontier non-terminal nodes of the d-tree description coupled with their corresponding concepts in the applicability semantics. In order for the syntactic structure of the mapping rule to be complete other d-trees have to be subserted at these nodes.

The *names* of MRs are used mainly for system internal purposes when the system is reporting what it is doing and what MRs it is attempting to apply.

The *applicability semantics* associated with a MR is used to license its application. The applicability semantics graph has a single node ("the semantic head") which acts as a root (graphically we indicate this concept by an arrow see Figure 5.7). The head annotations may be argued to impose a hierarchical structure on the graph but it is important to note that the input semantics does not contain such annotations.

The syntactic part of a MR is represented as a *d-tree*. The nodes in the d-tree are complex feature structures which by virtue of the unification operation used to combine nodes when trees are put together (using subsertion and sister-adjunction) can restrict the trees they can combine with.

Because of the larger domain of locality of d-trees we need an additional component which identifies which parts of the 'large' semantic structure correspond to which syntactic subdomain. Thus, in a transitive verb MR the generator must have a way of relating pieces of semantics with what are to be generated as a subject and the object. The *linking relation* identifies the head concept for a particular syntactic node.[9] It should be noted, however, that it is possible to have nodes which are not linked to the semantics, i.e., they 'do not have semantic content'. This is the way we treat particles and certain fixed phrases and idiomatic constructions. So the link relation viewed as a function from syntactic nodes to semantic concepts is a partial function. Graphi-

[8] What we call a linking relation here is very different from the `link` relation in, for example, left-corner parsing where the link relation is the transitive closure of the leftmost daughter relation between two (syntactic) category symbols.

[9] This is similar to LFG where nodes in the c-structure are linked to paths in the f-structure.

cally we use dotted lines to show which syntactic node is linked to which concept (see Figure 5.7).

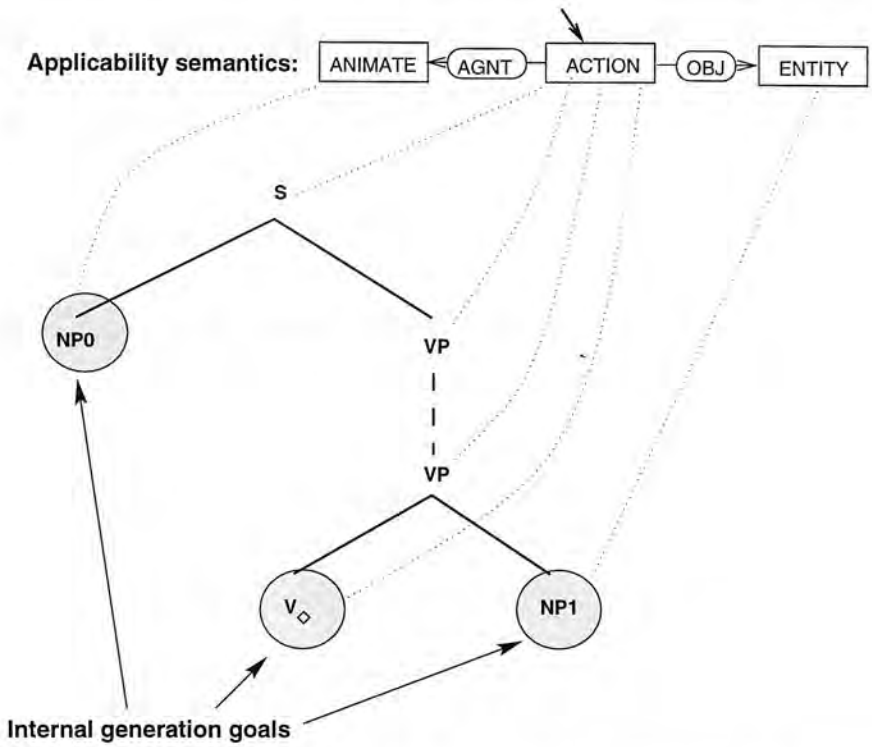


Figure 5.7: A mapping rule for transitive constructions

The *generation goals* of a MR refer to the non-terminal frontier nodes of the d-tree and associate them with a concept from the applicability semantics for the MR.[10] The generation goals also state what should be the head of the constituent denoted by this non-terminal in a complete derivation—this is done on the basis of the link relation. Lexical mapping rules have an empty list of generation goals. It is interesting to consider the question of temporal ordering of the list of internal goals, i.e., whether the generation goals should be kept in a list data structure and not a set, for example (the surface order of constituents is determined by the structure of the tree anyway). For historical reasons most generation systems attempt to generate constituents in the

[10] It is possible to define the generation goals as the set of non-terminal frontier nodes only. The corresponding semantics can always be determined from the linking relation. The reason we define generation goals the way we do is to have the same data structure for embedded generation goals at all levels.

order in which they appear in the surface structure (left-to-right for English[11]). For certain constructions and types of semantic analysis (e.g., the subject-head schema in HPSG) this appears problematic and this has led to the use of complicated delay mechanisms. The larger domain of locality that d-trees offer allows us to consider certain interactions explicitly. We generate syntactic heads first because they select for their arguments.

5.1.5.3 Meaning of a mapping rule

A MR can be interpreted as follows. A MR can license a derived (complete or partial) structure which includes the d-tree in the syntactic part of the MR. The semantic contribution of the MR is its applicability semantics minus the semantics of the generation goals (not necessarily a connected graph). The linking between syntactic nodes and semantic concepts remains invariant no matter how the mapping rule is used. The MR is a declarative device stating the relationship between syntax and semantics.

Figure 5.7 shows an example of a mapping rule for a transitive construction. The arrow indicates the head of the construction. We take up the issue of the head annotations in Section 5.5.1. The applicability semantics of the above mapping rule is: [ANIMATE]←(AGNT)–[ACTION]–(OBJ)→[ENTITY]. The internal generation goals (shaded areas) express the following:

1. generate [ACTION] as a verb and subsert (substitute, attach) the verb's syntactic structure at the V◇ node;
2. generate [ANIMATE] as a noun phrase and subsert the newly built syntactic structure of the subject at node NP_0; and
3. generate [ENTITY] as another noun phrase and subsert the newly built syntactic structure of the object at node NP_1.

After each subsertion the corresponding semantics of the current mixed syntactic-semantic structure is augmented (updated) by maximally joining it with corresponding

[11] For languages like Japanese generating constituents in a right-to-left fashion would make more sense.

semantics of the complement structure on the concept from the original applicability semantics as required by the linking relation of the top-level MR.

In previous publications [Nicolov *et al.* 95, Nicolov *et al.* 96] we have used a different notation for talking about MRs. While the formal notion of a MR is the same we believe the current notation is simpler. One aspect in which there is a difference is that in the former notation, where inner syntactic nodes were annotated with an instruction as to how the graphs of their constituents were put together, it was possible to interpret the MRs as specifications of how the input conceptual graph can be decomposed. In the former notation it was also possible to view a MR as the result of an evaluation of the semantic instruction associated with the top syntactic node in the tree description.

In the new formulation inner syntactic nodes are related (via the link relation) to nodes in the applicability semantics. We have done away with the instructions and produce the built semantics of the whole sentence as a combination (maximal join) of the applicability semantics of the applied mapping rules. Previously we calculated the built semantics at the end of the generation process while now we update it in an incremental fashion. Yet, the incremental way of calculating the built semantics was possible within the previous framework—our design decision then was different though.

5.1.5.4 The notion of consumption

The way we have defined MRs implicitly defines a notion of consumption, i.e., how much of the original input semantics is consumed. It is probably better to think about the semantic contribution of a mapping rule rather than what it consumes because we allow a mismatch between what has to be expressed and what can be expressed.

Lexical mapping rules (which do not have embedded generation goals) consume all of their applicability semantics, i.e., their contribution to the final built semantics is a copy of their applicability semantics. Certain concepts of the applicability semantics will be related to their counterparts in the input semantics or the lower semantic bound[12] (but these concepts will not be joined with the input semantics so that the

[12] Later we will formalise this notion of linking (see Section 5.10.3).

built and input semantic structures are not polluted).

The semantic contribution of non-lexical rules is slightly more complicated. It is the applicability semantics of the MR minus the nodes referred to by the internal generation goals. The head semantic node is excluded too if it is not associated with a syntactic node. This is the case for modification mapping rules which in their applicability semantics mention the semantic head node at which the modification semantics can be attached. We will see some examples which demonstrate the mechanisms later on. Strictly speaking what remains after excluding the above mentioned portions of the applicability semantics may not be a proper conceptual graph. This is why when we defined the meaning of a mapping rule above we discussed what the built semantics is *after* appropriate trees for the internal generation goals have been put into place (using subsertion).

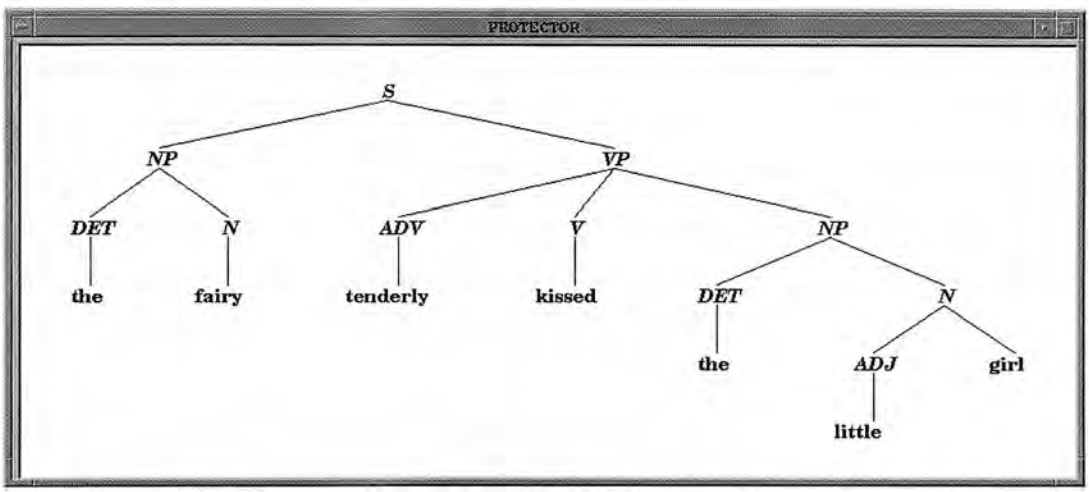


Figure 5.8: Example output tree

5.2 Outputs

5.2.1 Syntactic structure

The main output of any generation system is some text in a natural language. Our sentence generator produces an elaborate syntactic structure in the form of a d-tree. The leaf nodes of the tree are labeled with fully inflected lexical items.

Figure 5.8 shows the shape of a syntactic representation for the sentence *The fairy tenderly kissed the little girl.* The non-terminal nodes have more complex structure. In general all nodes in the tree are labeled with large feature structures.

5.2.2 Corresponding semantics and linking

An important output of the system is the semantics corresponding to the generated sentence (*BuiltSem*). This is a structure built compositionally using the applicability semantic structures of all MRs that have been applied—*BuiltSem* is the maximal join of the applicability semantics of all MRs that have been used to derive the current sentence. The conceptual graphs are not joined arbitrarily—the concepts of the internal generation goals are joined with the head concepts of the applicability semantics of the embedded generation goals (see Section 5.5.1 for description of head annotations).

In addition the generator also knows about the correspondence links between syntactic nodes of the final d-tree and their corresponding semantic head nodes in *BuiltSem* (such correspondences are established using the linking relation). This is of course vital when the output of the generator is used for other purposes than just displaying the final text or syntactic tree. For example if a speech synthesiser is to articulate the word *"wind"* it would need to know at least the syntactic category of the word. In some cases knowing the semantics is important too (indeed for this case too).

Having discussed the knowledge sources used in our generation system, the inputs and the outputs, we now turn to processing issues, i.e., we look at how these knowledge sources are actually used in the generation process. But before we start looking at different generation algorithms we first describe the declarative notion of derivation in our framework.

5.3 Top-down generation

In this section we describe one of the generation algorithms that is used in PROTECTOR. In Figure 5.9 and later in Figure 5.20, which illustrate some semantic aspects of the processing, we use a diagrammatic notation to describe semantic structures which are

actually encoded using conceptual graphs.

The input to the generator is *InputSem*, *LowerSem*, *UpperSem* and a mixed structure, *Partial*, which contains a syntactic part (usually just one node but possibly something more complex) and (possibly an empty) set of links between syntactic nodes and their semantic head nodes. Initially *Partial* represents the syntactic-semantic correspondences which are imposed on the generator. It has the format of a structure similar to the representation used to express mapping rules (see Figure 5.7). Later during the generation *Partial* is enriched and at any stage of processing it represents the current tree and syntactic-semantic correspondences.

We have augmented the DTG formalism so that the semantic structures associated with syntactic nodes will be updated appropriately during the subsertion and sister-adjunction operations. This process is constrained by the annotations for semantic heads (graphically indicated by arrows) and the notion of obligatory concepts as explained in Section 5.5.1.

Below we describe the top-down generation strategy. We have decided to look initially at top-down generation for a number of reasons:

1. From a psycholinguistic point of view generation is often claimed to be inherently a top-down process: IPF [De Smedt 90].

2. Historically most generators use such a strategy: FUF [Elhadad 93], LFG generators [Kohl 91], the top-down prediction in SHDG [Shieber *et al.* 90], the generator in the IDAS system [Reiter & Mellish 92], etc.

The stages of generation are:

1. building an initial skeletal structure;

2. attempting to consume as much as possible of the semantics not covered in the previous stage;

3. converting the partial syntactic structure into a complete syntactic tree; and

4. generation of follow-up sentences.

We consider each stage in turn.

5.3.1 Building a skeletal structure

Generation starts by first trying to find a mapping rule whose semantic structure matches[13] part of the initial graph and whose syntactic structure is compatible with the goal syntax (the syntactic part of *Partial*). If the initial goal has a more elaborate syntactic structure and requires parts of the semantics to be expressed as certain syntactic structures this has to be respected by the mapping rule. Such an initial mapping rule will have a syntactic structure that will provide the skeleton syntax for the sentence. If lexicalised DTG is used as the base syntactic formalism at this stage the mapping rule will introduce the head of the sentence structure—the main verb. If the rule has internal generation goals then these are explored recursively (possibly via an agenda—we will ignore here the issue of the order in which internal generation goals are executed). Because of the minimality of the mapping rule, the syntactic structure that is produced by this initial stage is very basic—for example only obligatory complements are considered. Any mapping rule can introduce additional semantics and such additions are checked against the lower semantic bound. When applying a mapping rule the generator keeps track of how much of the initial semantic structure has been covered/consumed. Thus at the point when all internal generation goals of the first (skeletal) mapping rule have been exhausted the generator knows how much of the initial graph remains to be expressed.

5.3.2 Covering the remaining semantics

In the second stage the generator aims to find mapping rules in order to cover most of the remaining semantics (see Figure 5.9). The choice of mapping rules is influenced by the following criteria:

Connectivity: The semantics of the mapping rule has to match (cover) part of the covered semantics and part of the remaining semantics.

[13] via the maximal join operation

Integration: It should be possible to incorporate the semantics of the mapping rule into the semantics of the current structure being built by the generator.

Realisability: It should be possible to incorporate the partial syntactic structure of the mapping rule into the current syntactic structure being built by the generator.

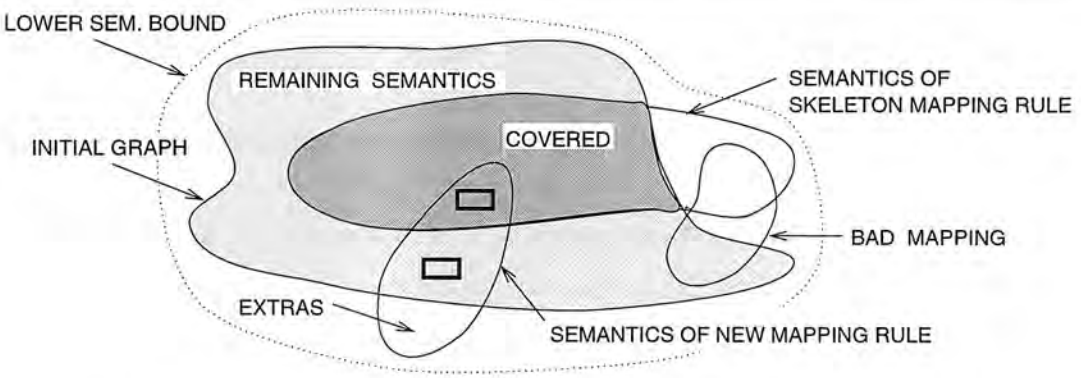


Figure 5.9: Covering the remaining semantics with mapping rules

Note that the connectivity condition restricts the choice of mapping rules so that a rule that matches part of the remaining semantics and the extra semantics added by previous mapping rules cannot be chosen (e.g., the "bad mapping" in Figure 5.9). While in the stage of fleshing out the skeleton sentence structure (Section 5.3.1) the syntactic integration involves subsertion, in the stage of covering the remaining semantics it is sister-adjunction that is used. When incorporating semantic structures the semantic head has to be preserved—for example when sister-adjoining the d-tree for an adverbial construction the semantic head of the top syntactic node has to be the same as the semantic head of the node at which sister-adjunction is done. This explicit marking of the semantic head concepts differs from [Shieber *et al.* 90] where the semantic head is a PROLOG term with exactly the same structure as the input semantics.

5.3.3 Completing a derivation

In the preceding stages of building the skeletal sentence structure and covering the remaining semantics, the generator is mainly concerned with consuming the initial semantic structure. In those processes, parts of the semantics are mapped onto partial

syntactic structures which are integrated and the result is still a partial syntactic structure. That is why a final step of "closing off" the derivation is needed. The generator tries to convert the partial syntactic structure into a complete syntactic tree. In practical terms this means that we have to eliminate the d-links by merging the node at the top and the node at the bottom of each d-link. Syntactically the feature structure annotations of the two nodes are unified. Semantically the head concepts corresponding to the nodes at top and bottom of a d-link are maximally joined.

5.3.3.1 Morphological processing

A morphological post-processor reads off the leaves of the final syntactic tree and inflects the words. Words are not inflected as soon as the lexical item is chosen because often the actual inflectional form depends on syntactic features which are not yet available. An example where this occurs in English is subject-verb agreement in declarative sentences in present tense. Following our strategy the generator first finds a lexical item for the verb. Yet, at this stage the number of the subject is not known (because it is not realised). Of course it is possible for the generator to choose nondeterministically one of the available forms. Because of coindexing between syntactic agreement features of the verb and the subject NP this will further constrain the generation goal for the subject NP. If the generator had chosen `number: plural` earlier and there are no plural realisations of the subject NP this leads to failure and the computations performed in the meantime are wasted. At the point when the verb is chosen not enough information is available and the best the generator can do is to take a 'blind leap'. While for English that causes little backtracking (because of its impoverished morphology) there are languages like Estonian which can have up to 196 inflected forms per base lexeme.

An alternative strategy that we have also considered is interleaving the morphological processing with the generation process but have the morphological component suspend unless the needed information is available, or there is only *one* fully inflected form. This latter condition is a form of expansion of deterministic goals (a functionality available in a recent formalism — CUF [Dörre & Dorna 93]). Committing to this single inflected form does not cause problems even if it instantiates further certain syntactic features. It

can help prune some branches of the search space. If backtracking occurs an alternative lexeme to the one that gave raise to the inflected form should be considered.

Both mechanisms have their merits. The first mechanism avoids a small amount of overhead while the second frees the developer from worrying about the actual order of execution of processes (morphological and syntactic decisions) and allows them to concentrate on the declarative aspects. The second alternative can be more relevant when the actual (inflected forms) words are needed as soon as possible, e.g., in the case of incremental generation, etc.

5.3.4 Top-down algorithm

In this section we specify algorithmically the top-down generation strategy.

---

global: *Sem*, *LowerSem*, *UpperSem*, *Cover*, *BuiltSem*, *Partial*;
type: *Sem*, *LowerSem*, *UpperSem*, *Cover*, *BuiltSem*: *cg_graph*;
Partial: *semantically annotated description*;

generate() : *syn_tree*; % inputs are *Sem*, *LowerSem*, *UpperSem* and *Partial*
begin
    Cover := *empty_graph*;
    BuiltSem := *empty_graph*;
    generate2(*Sem*, *Partial*);
    cover_remaining();
    close_derivation();
    return(*syn*(*Partial*))
end

Figure 5.10: Generation algorithm—top level

Sem is the initial semantics. It remains unchanged throughout the generation process.

LowerSem (*LowerSem*$\leq$*Sem*) is the lower semantic bound. Whenever the generator needs to say more information than is encoded in *Sem* then this is checked against the lower semantic constraint.

Partial is a tree description whose nodes are annotated with corresponding semantics. *Partial* can be updated—at any stage it contains the mixed syntactic-semantic structure generated so far. *Sem*, *LowerSem*, *UpperSem* and *Partial* are inputs to the algorithm.

Cover (a subgraph of *Sem*) represents how much of *Sem* has already been covered.

BuiltSem reflects the semantic structure of the string being produced (which is not necessarily identical to *Sem* or *Cover*).

Figure 5.11 describes the stage of finding an initial skeletal mapping and the stage of recursively exploring its embedded generation goals.

$syn(MS)$ and $sem(MS)$ are selectors—they give the syntactic/semantic structure of a mixed structure MS.

generate2(*GoalSem*, *GoalStr*);
type: *GoalSem* : *cg_graph*;
GoalStr : *semantically annotated description*;
begin
Find a mapping rule R s.t. :
$syn(R)$ is compatible with $syn(GoalStr)$ **and**
$sem(R)$ matches *GoalSem* % within *Sem*, *LowerSem* and *UpperSem*
BuiltSem := *merge*(*BuiltSem*, $sem(R)$); % join on the same concepts
Cover := *merge*(*Cover*, *max_projection*(*Sem*, $sem(R)$)) % $Sem \leq Cover$
Partial := integration of the mixed structure of R into *Partial*
for each internal generation goal $\langle IntSem, IntStr \rangle \in$ mapping rule R
do call *generate2*($IntSem$, $IntStr$)
end

Figure 5.11: Recursive descent processing of internal generation goals

---

A mapping rule R is simply an elementary d-tree whose nodes are annotated with appropriate semantics—a mixed structure. $syn(R)$ of a mapping rule R is only the syntax part of the mapping rule. $sem(R)$ of a mapping rule R is the semantics which acts as applicability condition for the mapping rule.

The function $merge(CG_1, CG_2)$ produces a join of the conceptual graphs CG_1 and CG_2 on a projection consisting of the concepts and relations that are the same in both graphs.

$IntStr$ in the internal generation goals is a usually a syntactic node (thus the generation goal will have to generate the $IntSem$ as a syntactic sub-tree conforming to this syntactic node). In general, however, $IntStr$ can be more complex—it is a semantically annotated tree description (mixed structure).

During the matching of graphs indices of concepts from $sem(R)$ become the same as indices of concepts in *Sem* (or *LowerSem*). This allows for the correct integration of the mapping rule's mixed structure in *Partial*—we know which concepts in *Partial* correspond to which concepts in the original semantics and can make sure that the integration of additional structures into *Partial* leads to the top syntactic node having the correct associated semantics.

---

cover_remaining();
begin
while $Sem \leq Cover$ (i.e., there is remaining semantics)
and there is a mapping rule R s.t. :
$sem(R)$ matches at least one concept in *Cover* **and**
$sem(R)$ matches at least one concept or relation in *Sem* but not in *Cover*
do apply mapping rule R % Integrate R in *Partial*; update *Cover* and *BuiltSem*
end

Figure 5.12: Covering the remaining semantics

Applying a mapping rule in Figure 5.12 means finding a syntactic node in *Partial* such that:

1. the node is a modification one (i.e., it can be modified)
2. semantically the mapping rule has the same head as the node in *Partial* being modified

Cover and *BuiltSem* will also need to be updated.

---

close_derivation();
begin
for each d-link $\in$ *syn*(*Partial*)
unify the *top* and *bottom* of the d-link
end

Figure 5.13: Closing a derivation

---

follow_up_gen();
begin
  while $Sem \leq Cover$ (i.e., there is remaining semantics)
    do
      Partial := sentence;
      generate2(*Sem*,*Partial*)
end

Figure 5.14: Generation of follow-up sentences

5.4 Example

In this section we illustrate how the algorithm works by means of a very simple example. Suppose we start with an initial semantics as given in Figure 5.15.

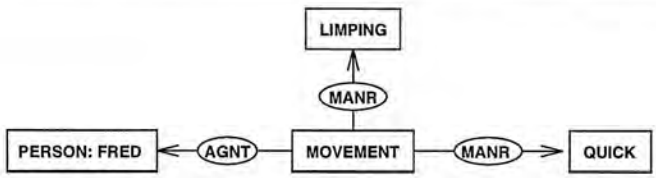


Figure 5.15: A simple conceptual graph

In the stage of building the skeletal structure mapping rule (i) in Figure 5.16 is used. Its internal generation goals are to realise the instantiation of [ACTION] (which is [MOVEMENT]) as a verb and similarly [PERSON:FRED] as a noun phrase. The generation of the subject noun phrase is not discussed here. The main verb is generated using terminal mapping rule[14] (iii) in Figure 5.16.[15] The skeletal structure thus generated is *Fred limp(ed).* (see Figure 5.17).

[14] Terminal mapping rules are mapping rules which have no internal generation goals and in which all terminal nodes of the syntactic structure are labelled with terminal symbols (lexemes).

[15] In Lexicalised DTGs the main verbs would be already present in the initial trees.

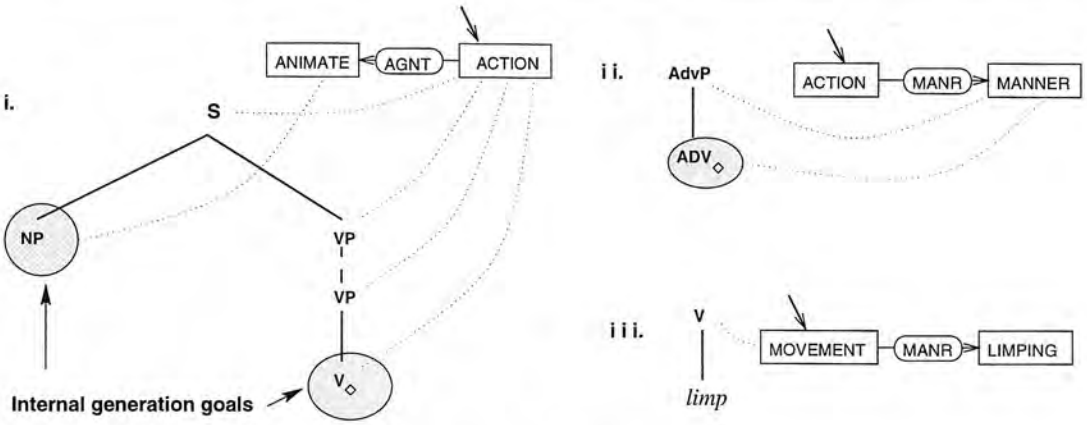


Figure 5.16: Mapping rules

An interesting point is that although the internal generation goal for the verb referred only to the concept MOVEMENT in the initial semantics, all of the information suggested by terminal mapping rule (iii) in Figure 5.16 is consumed. We will say more about how this is done in Section 5.5.

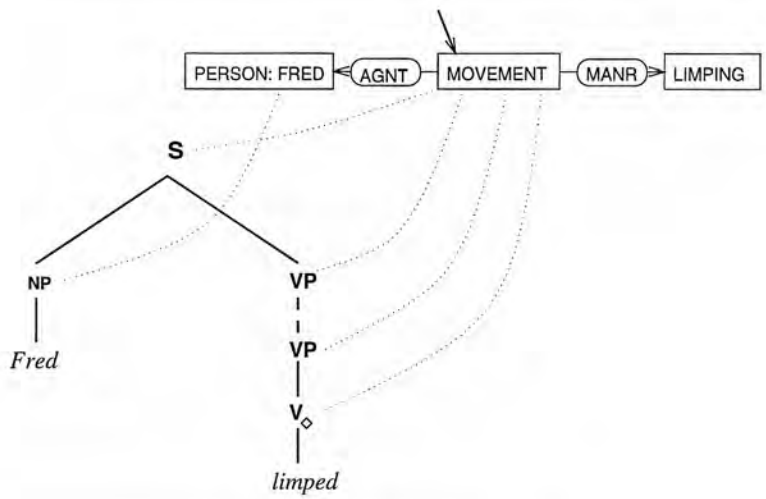


Figure 5.17: Skeletal structure

At this stage the only concept that remains to be consumed is QUICK. This is done in the stage of covering the remaining semantics when the mapping rule (ii) is used. This rule has an internal generation goal to generate the instantiation of MANNER as an adverb, which yields *quickly*. The structure suggested by this rule has to be integrated

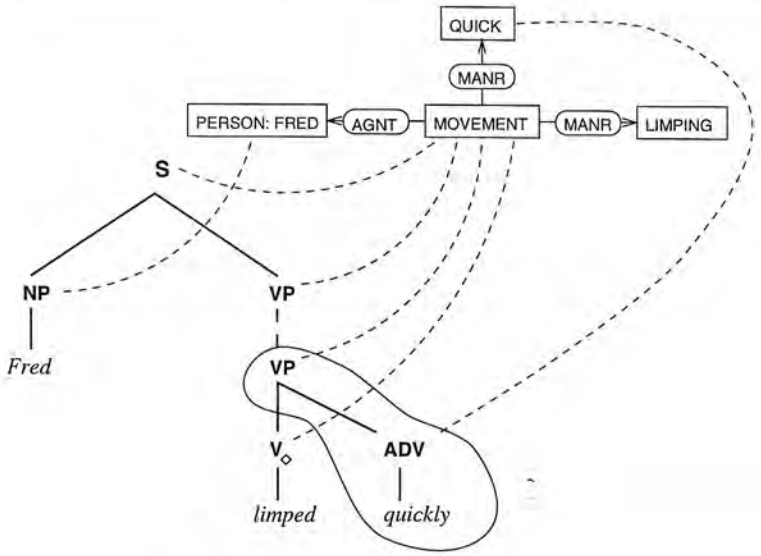


Figure 5.18: Final structure (d-tree)

in the skeletal structure. The generator has to find a node in the current structure which syntactically 'accepts' adverbial phrase modification and whose semantic head is MOVEMENT (i.e., the same as the instantiation of the head concept of the modification mapping rule). On the syntactic side the integration of both structures is done using sister-adjunction. The current mixed syntactic-semantic structure (which is in fact the last partial structure) is shown in Figure 5.18.

So far all of the input semantics has been consumed. No additional semantics has been introduced. In the syntactic part of the current partial structure there is one domination link. It can be removed by merging both VP nodes because the (syntactic) feature structures associated with them are unifiable[16] and the semantic heads of the two nodes are the same concept. After morphological post-processing the result is *Fred limped quickly* (see Figure 5.19).

The same input semantics, of course, can give rise to other sentences, for instance—

[16] For reasons of clarity we have only shown category information. The label of each non-terminal node is in fact a complex feature structure. In our simple example there is at least agreement information between the top VP node and the subject NP.

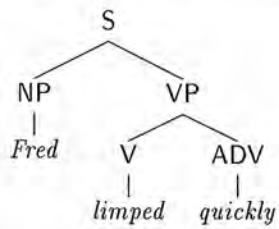


Figure 5.19: Final complete structure

Fred hurried with a limp.[17] This can be done by using a lexical mapping rule for the verb *hurry* which groups [MOVEMENT] and [QUICK] together and a *PP* expressing [LIMPING]. To get both paraphrases would be hard for generators relying on hierarchical representations.

5.5 Matching the applicability semantics of mapping rules

Matching of the applicability semantics of mapping rules against other semantic structures occurs in the following cases: (i) when looking for a skeletal structure; (ii) when exploring an internal generation goal; and (iii) when looking for mapping rules in the phase of covering the remaining semantics. During the exploration of internal generation goals the applicability semantics of a mapping rule is matched against the semantics of an internal generation goal. We assume that the following conditions hold:

1. The applicability semantics of the mapping rule can be maximally joined with the goal semantics (*GoalSem*).

2. Any information introduced by the mapping rule that is more specialised than the goal semantics (additional concepts/relations, further type instantiation, etc.) must be within the lower semantic bound (*LowerSem*). If this additional information is within the input semantics, then information can propagate from the input semantics to the mapping rule (the shaded area 2 in Figure 5.20). If the

[17] Our example is based on Iordanskaja *et al.*'s notion of maximal reductions of a semantic net (see [Iordanskaja & *et al.* 91, page 300]).

mapping rule's semantic additions are merely in *LowerSem*, then information cannot flow from *LowerSem* to the mapping rule (area 1 in Figure 5.20).

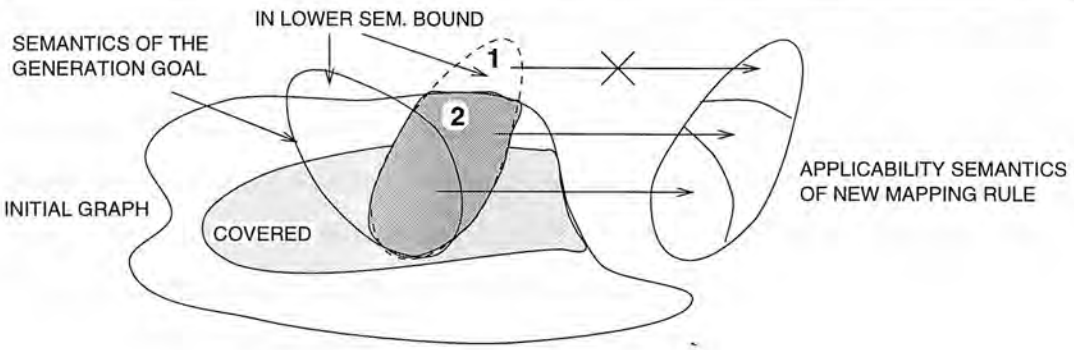


Figure 5.20: Interactions involving the applicability semantics of a mapping rule

Similar conditions hold when in the phase of covering the remaining semantics the applicability semantics of a mapping rule is matched against the initial semantics. This way of matching allows the generator to convey only the information in the original semantics and what the language forces one to convey even though more information might be known about the particular situation.

In the same spirit after the generator has consumed/expressed a concept in the input semantics the system checks that the lexical semantics of the generated word is more specific than the corresponding concept (if there is one) in the upper semantic bound (*UpperSem*).

5.5.1 Restricting the maximal join

We now consider two additional mechanisms which were mentioned earlier but so far we have avoided talking about them in order to present the notion of derivation in as simple and uncluttered a manner as possible. In this section we look at:

1. **headed conceptual graphs** in relation to the head annotations of the applicability semantics of MRs; and
2. **obligatory concepts** which should be in the maximal projection when the applicability semantics is matched with *Poll_In*.

The *head annotations* are used to restrict the MRs that are applicable to a certain generation goal. Normally when we have cases of generating [ENTITY] as an NP, MRs with applicability semantics containing a concept [ENTITY] will be activated. The condition that the semantics of the activated MRs should maximally join the goal semantics is rather weak. Thus, rules which we would not want to be activated will be considered too—for example constructions which will realise [ENTITY] as, say, an object of a subject-relativised transitive construction (see Figure 5.21) will be triggered too. If, however, the generation goal states that the semantics [ENTITY] is also annotated as a head and the subject-relativised transitive construction MR has the concept corresponding to the (empty) subject NP (i.e., the left [ENTITY] concept in Figure 5.21) as a head, and we require that maximal join aligned heads with heads, then we would avoid considering this rule and in general rules which will never appear in a derivation unless independently predicted.

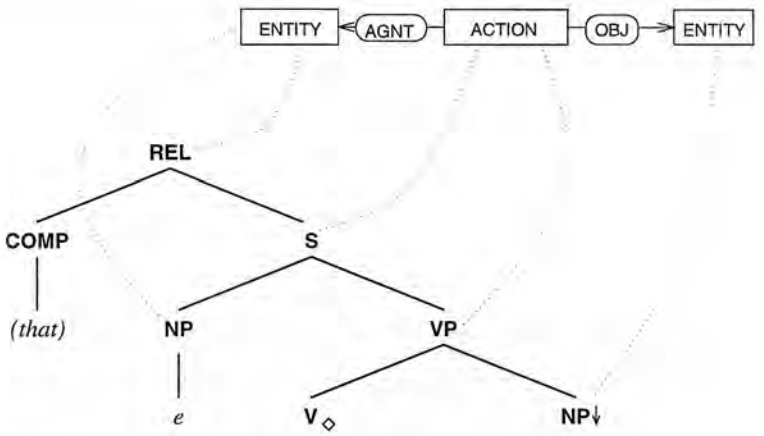


Figure 5.21: Subject-relativised transitive construction

When we discuss headed semantic structures from now on we will use an in-line notation which will mark head concepts with the symbol '⊙' and head concepts will look like this: [⊙ TYPE]. Thus, the applicability semantics of the MR in Figure 5.7 is actually: [ANIMATE]←(AGNT)–[⊙ ACTION]–(OBJ)→[ENTITY].

The set of *obligatory concepts* in the applicability semantics is yet another mechanism to constrain the paradigm so that the generator will not add arbitrary semantic information to the goal semantics. The set of obligatory concepts specifies which concepts

in the applicability semantics have to be in the maximal projection after the applicability semantics is maximally joined with the input semantics. MRs for relative clauses are again a case in point. If the generator has constructed a nominal phrase for a concept [PERSON] and considers whether additional information can be added to this nominal group then a MR for a relative clause will be applicable because the head of the relative clause semantics will also be the concept [ENTITY] and the heads will align in the matching. Yet, if the input semantics does not mention the concept [ACTION] which the MR for a relative clause adds it would not be reasonable to consider that rule. In order to block the application we introduce the mechanism of obligatory concepts. The MR for a relative clause has in its applicability semantics a set of obligatory concepts which includes the concept [ACTION]. The goal semantics will successfully join the applicability semantics of the MR only if the goal semantics contains the [ACTION].

Both headed annotations and obligatory concepts annotations directly address correctness—they prevent using a mapping rule when it is not desirable to use one. The generator is still complete in the sense that we do not miss valid derivations. These annotations greatly improve efficiency by radically reducing ambiguity.

5.6 Lexical choice

We view lexical choice to be interleaved with the syntactic processing. This implies that we do not expect our input to contain the lexical items specified in it.[18] Our method of performing lexical choice is an instance of the incremental consumption approach. The main idea is that words can have complex internal semantic structure. This allows one not to assume a perfect one-to-one correspondence between semantic primitives and lexical items.[19] The extent to which languages exploit this faculty of *packaging* semantic information in words varies from language to language.

Lexical items are chosen in the process of considering a mapping rule after the applicability semantics has matched the goal semantics. Then an appropriate word is chosen which can anchor the construction. In lexicalised DTGs each tree is assumed to

[18] As is the case with FUF and IDAS, for example.

[19] This contrasts with work on currently one of the largest generation systems—GIST [Not & Pianta 95].

already have an anchor in place. We do not store the combined elementary trees with their anchors as this will increase the size of the lexicon.[20] The important point is that the syntactic structure of mapping rules 'goes down' to a preterminal which provides sufficient information about the kind of lexical item that can be used. Semantically this preterminal is associated with a semantic concept which together with the input semantics (*InputSem*) helps guide the search for a word. *InputSem* is needed in case the lexical item introduces new semantic material. Thus our lexical chooser has as inputs:

1. a feature structure (syntactic constraints);
2. a (head) concept (headed conceptual graph in the general case); and
3. the input semantics (*InputSem*).

The lexicon consists of tuples of the form:

$$\langle CG_H,\ OCs,\ FS,\ LI \rangle$$

where CG_H is a headed conceptual graph; OCs are obligatory concepts (see Section 5.5.1); FS is a feature structure; LI is the lexical item (the base form of the word that will appear in the sentence).

A lexical entry $\langle CG_H,\ OCs,\ FS,\ LI \rangle$ is chosen if:

1. CG_H and $GoalSem$ match as headed conceptual graphs;
2. OCs in the maximal projection of the match of CG_H and $GoalSem$; and
3. The feature structure of the the goal unifies with FS—the feature structure of the lexical entry.

The entries in the lexicon can be indexed on the semantics and elaborate schemes for doing this in the context of conceptual graphs have been studied in [Ellis 95].

[20] This is not done for parsing with DTGs either.

A number of researchers have suggested that a non-hierarchical semantic formalism with a definitional mechanism (like conceptual graphs) is particularly good for lexical choice and the scenario that has been proposed is one where the initial semantics *InputSem* is transformed by a number of contraction steps which replace subgraphs by concepts that stand for them. A serious problem with such an approach is that even if contractions are lexically licenced (i.e., there is a lexical item that expresses the contracted subgraph) there is no guarantee that an arbitrary choice of contractions would yield a sentence that combines these items. Thus completeness is lost. In addition the subgraph covering problem is $\mathcal{NP}$-complete [Garey & Johnson 79].

5.6.1 Semantic look-up

In this section we present a method for finding lexical items assuming a certain indexing on the semantic structures.[21]

Let us consider the word *inflation*. Here is an example of how the complex concept of INFLATION can be defined using a neo-Davidsonian representation:

```
increase(E) & obj(E,P) & price(P) & manr(E,M) &
continuous(M) & loc(E,C) & country(C)
```

The semantic predicates can be ordered lexicographically:

1. `continuous(M)`, 2. `country(C)`, 3. `increase(E)`, 4. `loc(E,C)`,
5. `manr(E,M)`, 6. `obj(E,P)`, 7. `price(P)`

Thus if the generator has a subgraph that it needs to express it can find an appropriate lexical item (if there exists one) in logarithmic time. Though given a graph which can be expressed with n predicates in a quantifier free formula the number of possible subgraphs is 2^n (not all of which will be valid subgraphs).

[21] This method was suggested to us by Martin Kay in Bolzano 1993.

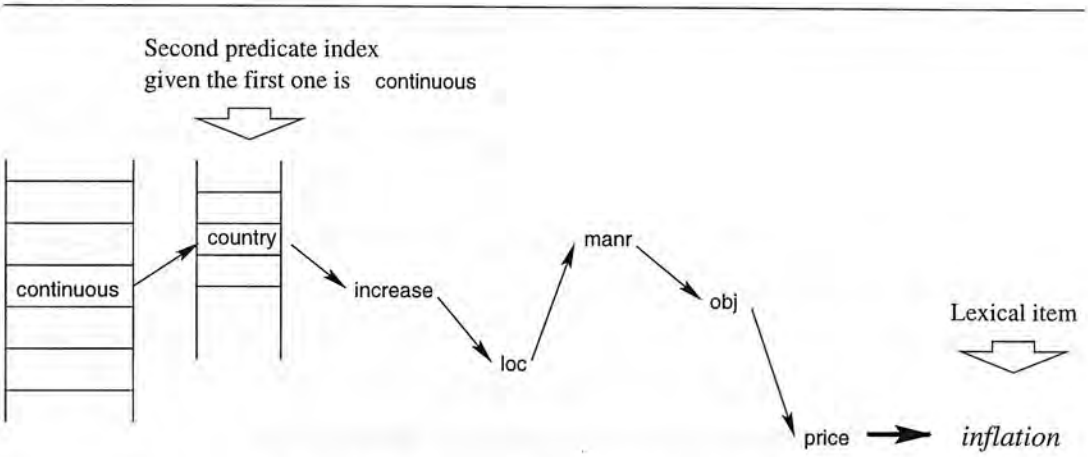


Figure 5.22: Semantic look-up of lexical items

5.7 Language specific ordering of modifiers

Our semantic representations do not assume the order in which modifiers are applied to be fully specified and where it isn't the generator uses language specific rules stating the order in which classes of modifiers can appear. In the context of multilingual generation for two different languages these modifier ordering rules can be different.

We define a partial ordering relation -<- between meaning classes corresponding to adjectives. The knowledge about the (partial) ordering between the semantic classes of modifiers is represented as the transitive closure of the -<- relation:

MATERIAL	-<-	NATIONALITY
AGE	-<-	MATERIAL
SIZE	-<-	AGE
PERSONAL_CHARACTERISTICS	-<-	SIZE
PRENOMINAL	-<-	*anything*

The meaning classes are part of the ontology; the additional mechanism we add is the ordering relation. The relation is transitive: given two modifiers of classes SIZE and MATERIAL the realisation of SIZE has to precede the realisation of MATERIAL (consider *big plastic box*; also because SIZE -<-* NATIONALITY *old Scottish whisky* is licenced but *Scottish old whisky* is rejected).

5.8 Approximate example

In this section we provide a worked example which indicates the effect of varying the distance between upper and lower semantic bounds given a fixed grammar, fixed semantic input, and a small set of mapping rules. Both qualitative (which?) and quantitative (how many?) effects on the set of available paraphrases should are covered.

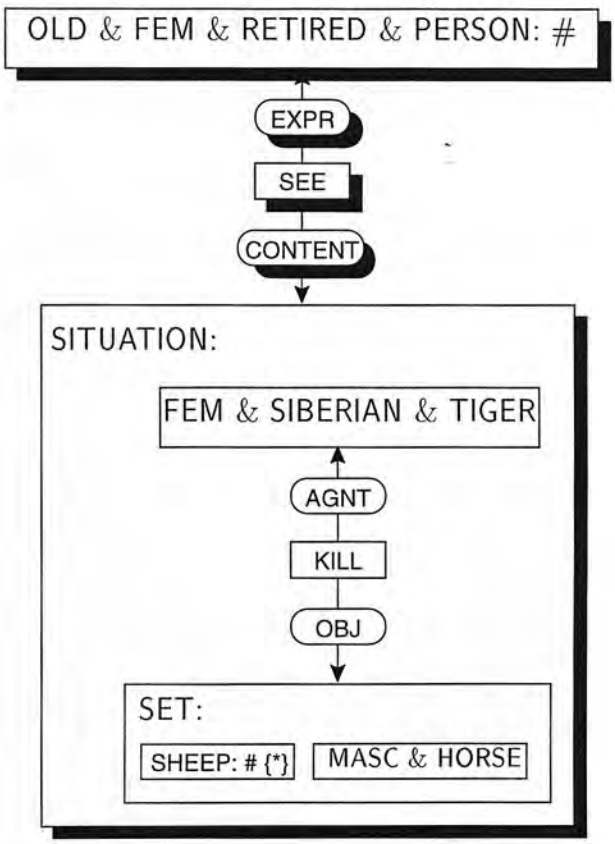


Figure 5.23: Input semantics for approximate generation example

Let us assume we are given input semantics semantics as shown in Figure 5.23. $C_1 \& C_2$ denotes the join of two the two concepts C_1 and C_2. In order to exemplify lower and upper semantic bounds let us also further assume that a concept is bounded above and below unless otherwise stated and let's have the following statements about the type of the concepts in the lower and upper semantics corresponding to concepts in the input.

Lower semantics ($Lower\_concept < Input\_concept$):

[OLD & FEM & RETIRED & PERSON & UNFRIENDLY] < [OLD & FEM & RETIRED & PERSON]

[FEM & SIBERIAN & TIGER] ≤ [FEM & SIBERIAN &TIGER]

(i.e., the generator is not allowed to further specialise this concept)

[FEM & SHEEP: # {*}@2] < [SHEEP: # {*}]

[SHIRE & MASC & HORSE] < [MASC & HORSE]

Upper semantics (*Input_concept* < *Upper_concept*):

[OLD & FEM & RETIRED & PERSON] < [OLD & PERSON]

[FEM & SIBERIAN & TIGER] < [TIGER]

[SHEEP: # {*}] ≤ [SHEEP: # {*}]

[MASC & HORSE] < [HORSE]

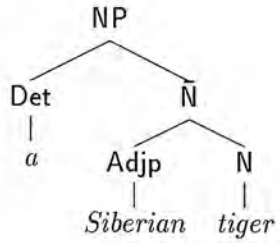
Also let us assume lexical mapping rules as in Table 5.2:

[**OLD**]	⇒	Adj:*old*
[FEM & **PERSON**]	⇒	N:*woman*
[FEM & **PERSON**]	⇒	N:*lady*
[RETIRED & **PERSON**]	⇒	N:*pensioner*
[OLD & FEM & **PERSON**]	⇒	N:*granny*
[OLD & FEM & **PERSON** & UNFRIENDLY]	⇒	N:*crone*
[**TIGER**]	⇒	N:*tiger*
[FEM & **TIGER**]	⇒	N:*tigress*
[**SIBERIAN**]	⇒	Adj:*Siberian*
[**SHEEP**]	⇒	N(num: sg|pl):*sheep*
[FEM & **SHEEP**]	⇒	N:*ewe*
[**HORSE**]	⇒	N:*horse*
[MASC & **HORSE**]	⇒	N:*stallion*
[**SHIRE**]	⇒	Adj:*shire*

Table 5.2: Example lexical mapping rules

We will view concepts like [SIBERIAN & **TIGER**] as a short hand notation for graphs like [SIBERIAN]←(ATTR)–[⊙ TIGER].[22] The reason why we want to have more structure is to be able to distribute/chunk the semantics appropriately when modifiers are used (cf. Section 5.4 for an example of how we constructed derivations including how modifiers are handled). Thus, the above graph can be expressed as:

[22] ⊙ is the annotation for a head concept.



Let us consider what possible paraphrases can be generated. These are sentences whose corresponding (shadow) semantics is within the lower and upper semantic bound. She should explicitly note that there are additional constraints on lexical choice (e.g., style) which we are deliberately ignoring here. In principle we could consider more complex examples; all that needed is that the semantic constraints allow for multiple solutions.

Given the provided constraints the following are valid paraphrases (among others):

1. *The old retired woman saw a Siberian tigress kill the sheep and a stallion.*
2. *The old woman saw a tiger kill the sheep and a horse.*
3. *The old lady saw a tigress kill the ewes and a stallion.*
4. *The pensioner saw a Siberian tigress kill the ewes and a stallion.*
5. *The old crone saw a Siberian tiger kill the two ewes and a horse.*

The choices for realisation of the different constituents are relatively independent and the number of possible paraphrases can be easily calculated:

OLD & FEM & RETIRED & PERSON & UNFRIENDLY: # ⇒ *the old retired woman*, *the old retired lady*, *the old woman*, *the old lady*, *the pensioner*, *the crone*, *the old crone*, *granny* (8 solutions)

SEE ⇒ *see*, *observe* (2 solutions)

FEM & SIBERIAN & TIGER ⇒ *a tiger*, *a Siberian tiger*, *tigress*, *a Siberian tigress* (4 solutions)

SHEEP: # {*} ⇒ *the sheep*, *the two sheep*, *the ewes*, *the two ewes* (4 solutions)

MASC & HORSE ⇒ *a stallion*, *a horse*, *a shire horse*, *a shire stallion* (4 solutions)

Overall there are $8 \times 2 \times 4 \times 4 \times 4 = 2^{10} = 1024$ possibilities. This number is not small and in the next two chapters we consider ways to efficiently compute alternative paraphrases (Chapter 6) and produce better paraphrases first.

On the other hand the model will rule out sentences like: *The person saw a tiger kill the sheep and a black Shire stallion* as in some parts this realisation overgeneralises the input semantics and in other parts it overspecialises it.

5.9 Generation of follow-up sentences

After the generator has completed the previous stages it still might be possible that there is still unexpressed semantics. It might be possible to realise the remaining semantics in a follow-up sentence and our generator attempts to do so. In principle natural languages are fairly flexible and can express complex ideas even in just a single sentence. In a generation system, however, the choices that have already been taken might not allow the incorporation of additional material (lexical gaps), or even if it were possible to extend the existing sentence this would result in clumsy, sometimes ambiguous constructions. In addition in a practical generation system it might not be possible to realise certain generation goals not because the language does not allow this but because the generation system lacks appropriate knowledge (rules) to express sentences of higher sophistication.

a paper by John (meaning *a paper written by John*)

If one wants to add that the paper has been edited (by someone else) then the construction:

a paper that has been edited by John

is misleading. It is better to generate something along the lines of:

... a paper by John ... The paper has been edited.

As Michael Elhadad notes in [Elhadad 93, page 102] the overall number of choices that the generator has to consider in generating an initial sentence and a follow-up sentence might be less than the choices needed to produce a single large sentence

on backtracking. Later in Chapter 6 we consider ways to improve this but from a psycholinguistic point of view it is also desirable to have the functionality of generating follow-up sentences. Currently the system has a simple module that does that — doing it properly requires incorporating more complex algorithms for generation of anaphors.

There are at least two possible approaches to expressing follow-up sentences:

1. Build a new semantic representation for the follow-up sentence by a process of impoverishing the current input semantics removing already expressed concepts but keeping the necessary minimum so that all unexpressed concepts are connected.
2. Mark the part of the graph that has been expressed and submit the 'marked' semantics as a new goal (the syntactic category being a sentence).

The second alternative is the one we have chosen. It offers the advantage that the generator need not worry whether it has removed expressed concepts that might be needed for triggering a more appropriate mapping rule. Both approaches need to distinguish between expressed and non-expressed concepts. The generator needs to keep track of additional data structures like local focus and global focus [Grosz & Sidner 86].

5.10 A derivation in our model

In this section we examine the logical relationship between semantic structures and their possible counterpart syntactic structures. We first formally define some notions about which we have talked informally in the preceding sections (mapping rules, linking relation). Then we will describe the class of legal partial trees and the class of legal complete trees. A derivation is defined in terms of partial and complete trees.

Definition 5.5 (Mapping rule)

A mapping rule is a tuple $MR = \langle CG, DTr, Link, Goals \rangle$ *where:*

- CG *is a conceptual graph (applicability semantics);*
- DTr *is d-tree description (syntactic part of the mapping rule). The nodes of the d-tree are typed feature structures;*
- $Link$ *is a 2-place relation linking nodes in the d-tree to their semantic head nodes in the applicability semantics:* $Link \subseteq nodes(DTr) \times concepts(CG)$ *(Link is a partial function in one direction* $Link : nodes(DTr) \rightarrow concepts(CG)$ *);*
- $Goals$ *is a list of frontier non-terminal nodes from the d-tree description* DTr *all of which are linked to nodes in the applicability semantics* CG *(i.e.,* $\forall\ Nd \in Goals\ \exists\ \langle Nd, Concept \rangle \in Link$*). Goals is the list of internal generation goals.*

A mapping rule for which $Goals = empty$ *(is an empty list) is called* ***lexical****. Otherwise it is called* ***non-lexical****.*

Definition 5.6 (Augmented DTG)

An augmented DTG *is a four tuple* $G = (V, T, MRs, S)$*, where* V *and* T *are a finite set of non-terminal and terminal symbols,* $S \in V$ *is a distinguished non-terminal (starting symbol) and* MRs *is a finite set of mapping rules whose syntactic structures are d-tree descriptions (whose non-terminal and terminal nodes are elements of* V *and* T *respectively).*

Definition 5.7 (Legal partial structure)

A legal partial structure is:

1. *a copy of a mapping rule, or*
2. *a structure which is the result of combining two legal partial structures as described by schema one and two below.*

The conceptual graph related to a legal partial structure is referred to as the ***corresponding graph****. In the case of copies of* MRs *this is a copy of the applicability semantics.*

A legal partial structure (whose syntactic part is a description of a d-tree) might be converted to a structure whose syntactic part which is a d-tree in which case the result will be referred to as a ***legal partial tree****.*

When describing the two schemata we will use the following notation for the annotations of the nodes — Syn/Sem where Syn will represent the syntactic label of the node (a feature structure) and Sem will be the concept in the corresponding semantics

to the structure which is associated with the node in question.[23] Formally Syn/Sem means that $\langle Syn, Sem\rangle \in Link$.

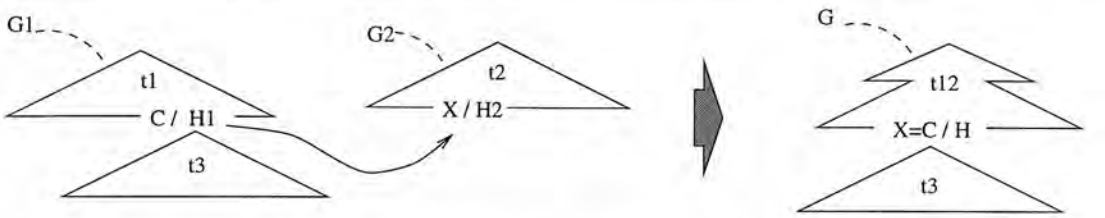


Figure 5.24: Schema one: augmented subsertion

Schema one. Given a legal partial (complement) structure $\langle G_1, DTr_1, Link_1, Goals_1\rangle$ with a maximal projection node in the syntactic part, annotated with C/H_1, and another legal partial structure $\langle G_2, DTr_2, Link_2, Goals_2\rangle$ with a syntactic frontier node annotated with X/H_2, then it is possible to construct a new legal partial structure $\langle G, DTr, Link, Goals\rangle$ for which:

- G is the maximal join of G_1 and G_2 where concepts H_1 and H_2 are identified;
- DTr is the result of the subsertion of description DTr_1 into description DTr_2;
- $Link = Link_1 \cup Link_2$ bearing in mind that the syntactic nodes C and X have been identified (unified) as well as the fact that semantic nodes have been joined (as a result of the maximal join of G_1 and G_2);
- $Goals = (Goals_1 \cup Goals_2) \setminus \{X\}$ (note that C and X have been identified).

If either of the first two conditions fails, so does the whole operation.

Figure 5.24 illustrates this schema. The tree t_{12} represents the tree t_2 with the component t_1 (which can be empty) integrated in it.

Schema two. Given a legal partial structure $\langle G_1, DTr_1, Link_1, Goals_1\rangle$ with an internal nonterminal node annotated with X/H_1, and a legal partial structure

[23] A similar notation was used for the SHDG algorithm.

$\langle G_2, DTr_2, Link_2, Goals_2 \rangle$ with a top node (maximal projection) annotated with M/H_2, it is possible to derive a new structure $\langle G, DTr, Link, Goals \rangle$ where M is sister-adjoined at X if (i) X's SAC (sister-adjunction) constraints allow the sister-adjunction of M and (ii) G_1 and G_2 can be joined at concepts H_1 and H_2.

- The semantics of the corresponding structure G, is the maximal join of G_1 and G_2 where concepts H_1 and H_2 are identified,
- DTr is the result of the sister-adjunction of description DTr_2 into description DTr_1;
- $Link = Link_1 \cup Link_2$ bearing in mind that nodes X and M have been identified (semantic nodes have been joined during the maximal join of G_1 and G_2);
- $Goals = Goals_1 \cup Goals_2$.

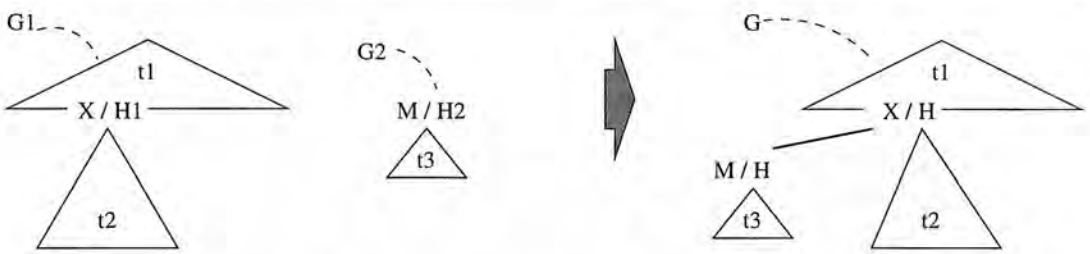


Figure 5.25: Schema two: augmented sister-adjunction

Figure 5.25 illustrates this schema. H is the join of the concepts H_1 and H_2.

Definition 5.8 (Legal complete structure)

A legal complete structure is triple $\langle CG, Tree, Link \rangle$ derived from a legal partial structure $\langle CG, TreeDescr, Link, empty \rangle$ (with an empty list of generation goals) by converting the d-tree description TreeDescr into a d-tree and then identifying (merging) the dominating and dominated nodes of the d-links in the d-tree. The resulting syntactic structure no longer contains d-edges and is the complete tree Tree. The following conditions must be satisfied:

1. *The syntactic labels of the nodes that are merged should be unifiable;*
2. *The two syntactic nodes N_1 and N_2 that are merged should have the same head concept, i.e., $\langle N_1, Concept_1 \rangle \in Link \wedge \langle N_2, Concept_2 \rangle \in Link \rightarrow Concept_1 = Concept_2$.*

Please note that the trees of legal complete structures do not contain frontier non-terminals. This is because legal complete structures are derived from legal partial structures with an empty list of generation goals. Thus, legal complete structures cannot be used in subsertion operations as the structure that is being subserted into.[24] Yet, it is possible to sister-adjoin a structure in a legal complete structure. In the sister-adjunction case, however, it is not beneficial to think of the complete structure as such. We will tend to avoid using combinations involving legal complete structures and will reserve the term for the final product of the generator. The outputs that we expect from the generator: syntactic tree, corresponding semantics (*BuiltSem*), and the linking between syntactic nodes to semantic nodes are exactly what a legal complete partial structure is all about.

Definition 5.9 (Augmented DTG derivation)

An augmented DTG derivation is a sequence of applications of schema one or schema two using any two legal partial structures and adding the resulting structure to the set of current legal partial structures. The final result must be a legal partial structure which can be turned into a legal complete structure.

Initially a derivation may start by combining two legal partial structures which are copies of generic MRs. Note that nothing requires that the initial generic MRs be lexical!

Augmented DTG derivation is the most general notion of derivation. There can be a number of different other notions of derivation which can be more intuitive. Here is one such notion:

Definition 5.10 (Spine DTG derivation)

A spine DTG derivation is an augmented DTG derivation with the additional requirement that one of the trees used in a combination operation be the result of the previous combination.

We talk more about (special kinds of) spine DTG derivations in Section 5.10.1.

[24] Though it is possible for a structure to be both a legal partial structure and a legal complete structure. The structure is partial in the sense that it might be further augmented (by sister-adjoining modification trees) and the structure is complete in the sense that it does not have explicit requirements for other structures to be incorporated into it.

In the PROTECTOR generation system we use spine derivations and in addition require that all subsertion operations are performed before any sister-adjunctions are attempted.

Definition 5.11 (Combination-ordered DTG derivation)

A combination-ordered DTG derivation is an augmented DTG derivation in which combinations of structures using subsertion for the syntactic parts of the structures are carried-out before combinations based on sister-adjunction.

These types of derivation are equivalent to augmented DTG derivation in the sense that for every augmented DTG derivation there are spine DTG derivations and combination-ordered DTG derivations yielding the same result. This follows from the fact that:

1. combining syntactic nodes using typed feature structure unification is order independent (the final result does not depend on the order in which things were combined), and
2. combining semantic structures using maximal join is likewise order independent.

It is interesting to point out that there are two syntactic operations (subsertion and sister-adjunction) which correspond to only one operation on the semantic side—maximal join.[25]

5.10.1 Top-down derivation

In this section we define top-down derivation, cast top-down derivation as a rewriting system and discuss its properties; we also give an example of top-down derivation. The general scenario will be taking an initial partial structure and refining it gradually until we reach a structure which can be converted to a complete structure and the semantics has been expressed. Top-down derivation differs from an augmented DTG derivation where there is a set of legal partial structures and new structures are being added to it[26]. Later in Section 5.10.2 when we define bottom-up derivation we will make use

[25] Though current revisions of DTG are considering just one operation on the syntactic side too—identifying two nodes in two tree descriptions.

[26] In the sense that top-down derivation is a very special kind of augmented DTG derivation.

of the more geneal notion of augmented DTG derivation which relies on sets of partial structures.

Definition 5.12 (Top-down derivation)

A top-down derivation is a combination-ordered, spine DTG derivation whose initial description is the generation goal and intermediate structures are not substituted or sister-adjoined in other legal partial structures.

Put in plain words a top-down derivation starts off with the generation goal. Then an initial mapping rule is chosen which provides the top-level syntactic structure for the realisation. Each of the frontier non-terminals of this initial construction is expanded in turn. In this way the complementation frame is built up in a top down manner. Note that the top-down derivation implies that the derivation tree is built top-down but *not* the actual derived tree. Because of the way extraction is handled, for example, the derived tree would not grow in a top-down manner as it would in the context-free grammar case. After the complementation stage there is a modification stage (thus the derivation is combination ordered). In the modification step the unconsumed semantics is consumed by using mapping rules which introduce different kinds of modifiers.

At first glance it might seem that combination-ordered, spine DTG derivation does not constrain the derivation very much. Thus there are a lot of generation strategies that fall in this class. That is of course true and the key point here is that the derivation is required to (i) start off with the initial generation goal; and (ii) intermediate structures are not allowed to be 'added to other new structures'. Both these conditions guarantee that the derivation structure will grow top-down.

Schemata one and two can be used to define the derives relation ($\Rightarrow$) in the following manner (each schemata corresponding to each definition below). The definitions make use of some notation which is explained in the paragraphs immediately after them:

Definition 5.13 (Directly derives)

1. $Descr[X\downarrow] \Rightarrow [Descr \cup TrD] \bullet \{X/proj\_node(TrD)\}$
2. $Descr[X*] \Rightarrow Descr \cup ModTrD \cup idom(X*, root(ModTrD))$

Clause 1 above corresponds to schema 1 and describes how a description can be rewritten using subsertion. The clause states that a description *Descr* which has a substi-

tution node X↓ can be rewritten as (derives) a new description which is the result of subserting an elementary (complement) description *TrD* by equating X↓ with a possible projection node from *TrD* (*proj_node*(*TrD*)). In the resulting description which is the union of both descriptions (*Descr* and *TrD*) *X* is substituted uniformly for the node *proj_node*(*TrD*).[27] In our implementation we unify the substitution node X↓ and the projection node of the complement structure *proj_node*(*TrD*). The definition of $\Rightarrow$ does not impose specific requirements about which substitution node in *Descr* or which of the projection nodes of *TrD* are picked. In PROTECTOR we consider leftmost substitution nodes first.

Clause 2 above corresponds to schema 2 and describes how a description can be rewritten using sister-adjunction. The clause states that a description *Descr* which has a modification node (X*)[28] can be rewritten (derives) a new description which is the result of sister-adjoining a (modification) description *ModTrD* by adding the constraint that X* immediately dominates the root *Root* of *ModTrD*: *idom*(X*, *Root*).

Schemas one and two have two descriptions on their lefthand-side (LHS) while the LHS of $\Rightarrow$ contains only one description. The second description is still there—we have just written it on the other side of the $\Rightarrow$ sign.

Both clauses that we give above are the counterparts of "$\alpha A\beta \Rightarrow \alpha\gamma\beta$ if $A \rightarrow \gamma$ is a production in a CFG".

Having defined $\Rightarrow$ we can define the relation derives in k steps ($\overset{k}{\Rightarrow}$) by the following inductive definition:

Definition 5.14 (Derives in k steps)

1. *0 step derivations:* $\forall D.D \overset{0}{\Rightarrow} D$
2. *1 step derivations:* $D_0 \overset{1}{\Rightarrow} D_1$ *iff* $D_0 \Rightarrow D_1$
3. *k step derivations:* $D_0 \overset{k}{\Rightarrow} D_k$ *iff* $D_0 \Rightarrow D_1$ *and* $D_1 \overset{k-1}{\Longrightarrow} D_k$

---

[27] The notation $Expr \bullet \{X/Y\}$ for expressing that X is substituted for Y in expression $Expr$ is often used in theorem proving.

[28] We use the * to annotate modification nodes by analogy with TAGs where foot nodes of auxiliary trees were marked with a '*' and these were the nodes in the auxiliary tree that are identified with the node that is being modified.

Definition 5.15 (Derives in zero or more steps)

$$Descr \stackrel{*}{\Rightarrow} D \quad iff \quad \exists k \geq 0.\ Descr \stackrel{k}{\Rightarrow} D$$

Simply speaking $\stackrel{*}{\Rightarrow}$ is the reflexive, transitive closure of $\Rightarrow$.

Top-down derivations are also combination-ordered ones so we can write a top-down derivation in the following way:

$$\underbrace{D_0 \Rightarrow D_1 \Rightarrow \ldots \Rightarrow D_j}_{\text{complementation stage}} \ ; \ \underbrace{D_j \Rightarrow \ldots \Rightarrow D_n}_{\text{modification stage}}$$

5.10.1.1 Derivations, rewrite systems and termination

It was not a coincidence that in the preceding section we referred to the 'derives' relation also as 'rewrites'. Indeed we can view derivations as a chain of rewritings of expressions and thus view generation as a special case of a rewrite system.

Unlike most rewrite systems, in generation we want to allow one expression to be rewritten in a number of possible final expressions (realisations), as opposed to having a single normal form in which the initial expression will be rewritten.

Let us take a closer look at termination. We want starting from an initial expression to be able to rewrite it into a final form in a *finite* number of steps. Here is clause 1 of the definition of $\Rightarrow$ again:

$$Descr[X\downarrow] \ \Rightarrow \ [Descr \ \cup \ TrD] \bullet \{X/proj\_node(TrD)\}$$

A substitution node is by definition a non-terminal node at the frontier of a d-tree (or a description of a d-tree). If a node is subserted into, it can no longer continue to be a substitution node because it will inherit the child nodes of the projection node of the root of the component that was substituted at the substitution node. (Note that the fact that $X\downarrow$ ceases to be a substitution node is not obvious from the notation!) Thus a substitution node can be subserted into only once. On the other hand a single substitution node is 'replaced' by all the substitution nodes of the d-tree that was subserted (in terms of the overall number of substitution nodes before and after the

subsertion). Yet, when lexical mapping rules are used the number of substitution nodes is strictly decreasing (by one). The scenario is parallel to goal introduction in PROLOG.

Let's examine clause 2 of the definition of $\Rightarrow$

$$Descr[X*] \Rightarrow Descr \cup ModTrD \cup idom(X*, root(ModTrD))$$

If we consider natural language examples involving the use of this rewrite rule, we find that the modification node X* continues to be such after the rewriting. Even worse it can be modified by the very same modification d-tree $ModTrD$:

1. *a long, long story*
2. *a very, very civilised way* [*of handling the situation*]

Arguably these phenomena are constrained, yet what will guarantee us termination will be the resource-based notion of consuming the semantics. Note how our definition of $\Rightarrow$ used only the syntactic side of mapping rules. We study the way the semantics constrains the derivation in the next section.

5.10.1.2 The first step

What is the nature of the first operation that gets us from the initial description to the first description? Is it in some way different from any of the other steps that are made in a derivation?

Standardly in generation the initial goal is a statement "realise the following semantics *Sem* as a syntactic structure (tree) whose top level category matches *Category*". In our framework we can do that very easily by stipulating that the root of the initial d-tree unifies with *Category*.

In the general case when we have a more elaborate description we require that the initial description 'subsumes' the final tree (or even the last d-tree). Note that given the generality of augmented DTG derivation it is not possible to impose additional constraints on the first step. For example consider a case where the initial description

describes the top of the final tree and the first step starts off by choosing a d-tree which will be deeply embedded in the final derived tree.

In top-down derivation we can impose tighter constraints between the initial description and the first d-tree—they have to match *top-down*.

5.10.1.3 Example of top-down derivation

We give an example of the top-down derivation for the sentence *The fairy tenderly kissed the little girl.*[29] We use indices to show at which step certain parts of the syntactic structure were added to the current realisation. These should not be confused with the numbers of complements that we used in Chapter 4.

The derivation starts off with the initial goal S_0. In the first derivation step an initial skeleton mapping rule is chosen whose syntactic part is shown as the second tree in Figure 5.26. The anchor for that mapping rule (and the corresponding d-tree description) is *kiss-ed*. This particular mapping rule has two embedded generation goals (one for generation of the subject and one for generation of the object). We use an ordering function that chooses the left-most goal as the next goal to work on. In the next (second) step a skeleton mapping rule (anchored by *fairy*) for the subject noun phrase is chosen which has its own embedded goal to generate the determiner (which happens in the third derivation step) and more structure is added to the current d-tree description. Similarly the object noun phrase is generated in the fourth and fifth steps. This concludes the complementation stage. The input semantics still contains structure which is not covered corresponding to the modifiers *tenderly* and *little*. In the modification stage the system chooses a mapping rule that covers some of the unexpressed semantics and in this case it first realises *tenderly* as an adverb in the sixth derivation step. Note how the modification (sister-adjunction) happens at the bottom of the d-link (within a d-tree description). Afterwards in the seventh step *little* is realised as an adjective. At present in the modification stage the system chooses mapping rules non-deterministically. The final partial syntactic structure (description) needs to be converted into a complete structure by merging the top and bottom nodes of d-links. The resulting tree is shown in Figure 5.27.

[29] Due to space constraints (as to what we can put in a one page figure) we do not show the semantics.

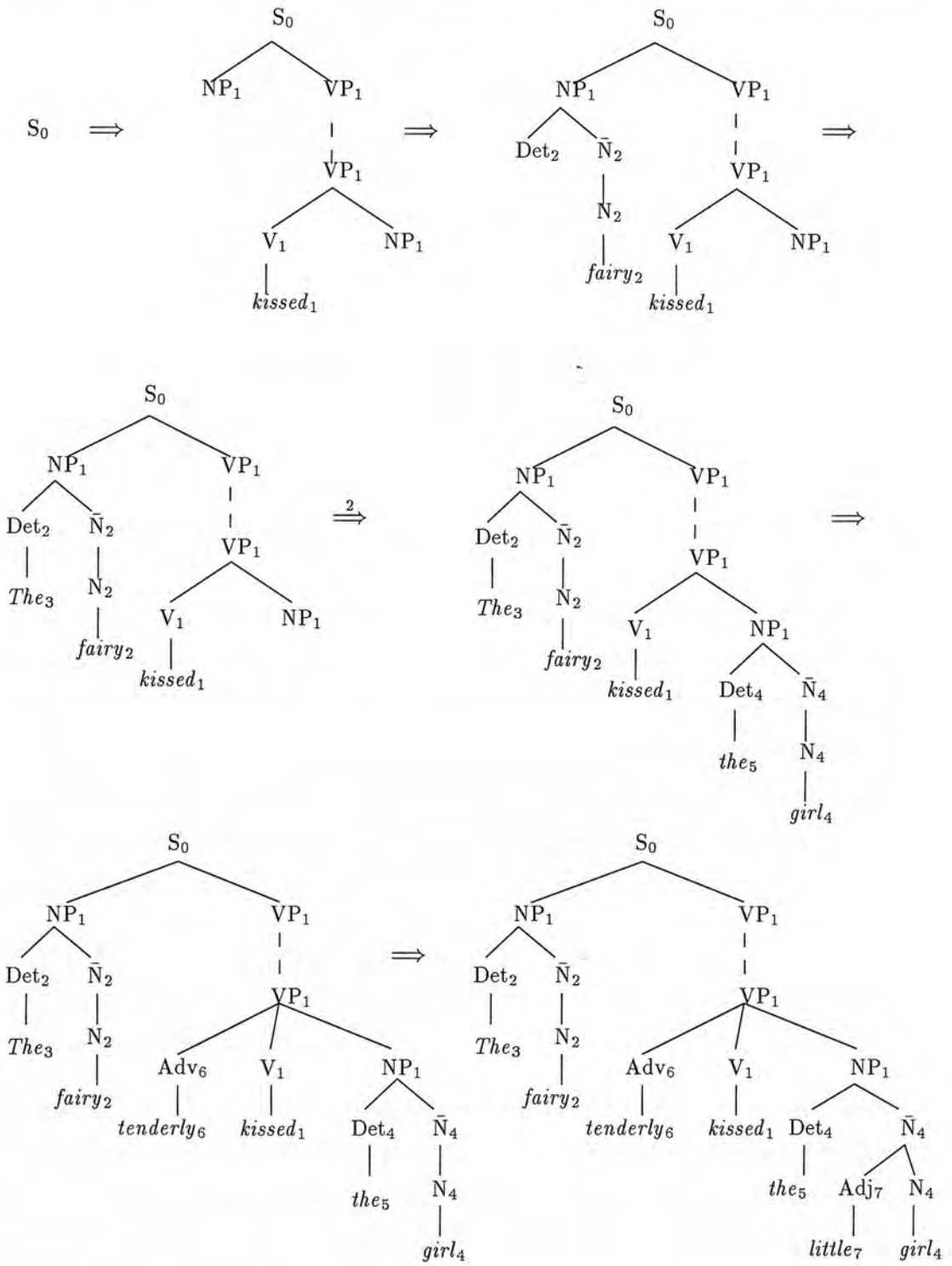


Figure 5.26: Top-down derivation (d-trees viewed as descriptions)

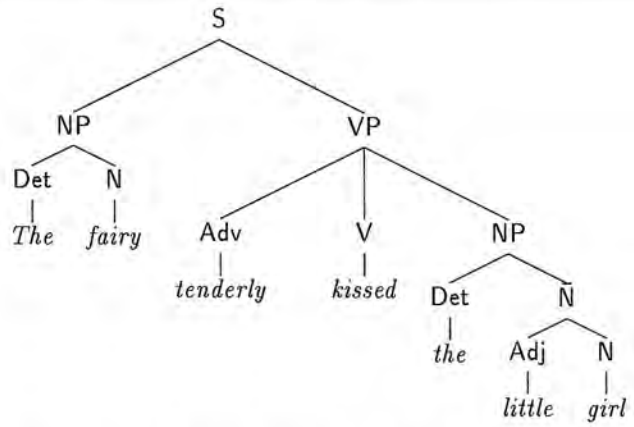


Figure 5.27: Top-down derivation result (derived tree)

5.10.2 Bottom-up derivation

In this section we describe bottom-up derivation. By doing this we aim to show the generality of our framework and to stress the declarative nature of the knowledge sources used by the generator. Note that none of the current generation systems runs in alternative modes and there are no studies that have concentrated on evaluation of different generation strategies. We do not do this but the declarative specification of derivation and the description of alternative strategies (top-down and bottom-up) is a step in that direction.

Why is there a need to define bottom-up derivation given that we have defined what an (augmented) DTG derivation is already? And why is it not possible to re-use the top-down definition somehow?

Augmented DTG derivation is rather general and we want to know what constraints on the processing will give a bottom-up regime. As for re-using the top-down definition of derivation, it is possible to define bottom-up processing as coming up with a top-down derivation *in reverse*.[30] Unfortunately that doesn't really tell us how we can build

[30] Such glossing over what bottom-up processing is is common in formal language theory:

> ... and if the substring is chosen correctly a each step, a rightmost derivation is traced out in reverse. [Aho *et al.* 86, 195]

generators that work *forwards* in a bottom-up regime.

Definition 5.16 (Bottom-up derivation)

A bottom-up derivation is an augmented DTG derivation for which the initial set of partial legal structures is the set of mapping rules that match the input semantics.

Initialisation. For all lexical MRs that match *InputSem* derive structures $\mathcal{T}_0$ corresponding to them.

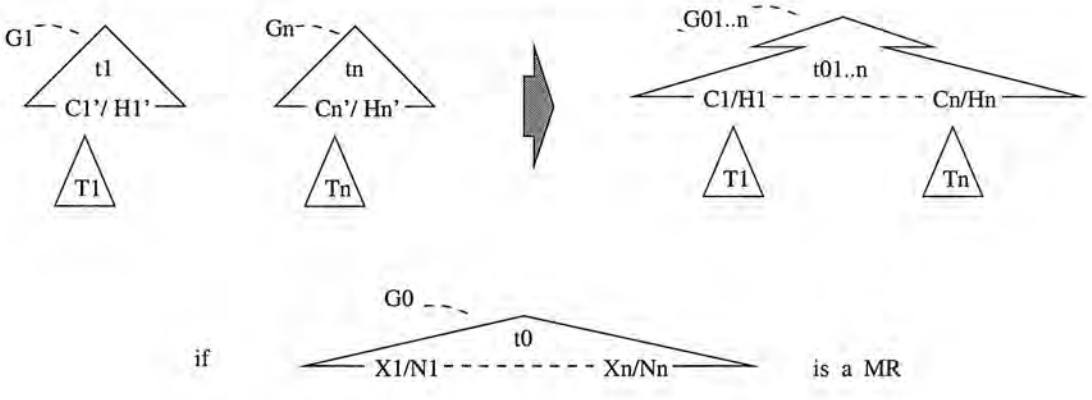


Figure 5.28: Schema one for bottom-up generation

Schema one. Given n syntactic (complement) structures with a top node (maximal projection) annotated with C_i/H_i and corresponding semantics G_i it is possible to construct a new syntactic structure as a result of a subserting all existing complement structures into the syntactic part of an existing MR if: the applicability semantics of MR can be maximally joined with $InputSem$, G_i can be maximally joined with the corresponding goal semantics of MR.[31] The semantics associated with the resulting structure, $G_{01...n}$, is the join of the instantiated G_0 and G_1, ... , G_n.

Figure 5.28 illustrates this schema. The tree $t_{01..n}$ represents the tree t_0 with the components t_1, ..., t_n (which can be empty) integrated in it.

[31] For the moment we do not allow MR to introduce extra goals which are outside of *InputSem* even though these might be within *LowerSem*. Such goals would either have to be explored top-down or they would require a mechanism of augmenting the Input semantics. The second option might lead to unwanted interactions between structures based on the augmented semantics and other structures.

Let $\mathcal{T}_i$ be the union of $\mathcal{T}_{i-1}$ and the set of all combinations of complement structures $T_1, T_2, \ldots, T_n$ (such that $T_k \in \mathcal{T}_{i-1}$ for $1 \leq k \leq n$) in an initially chosen mapping rule $T \in \mathcal{T}_0$.

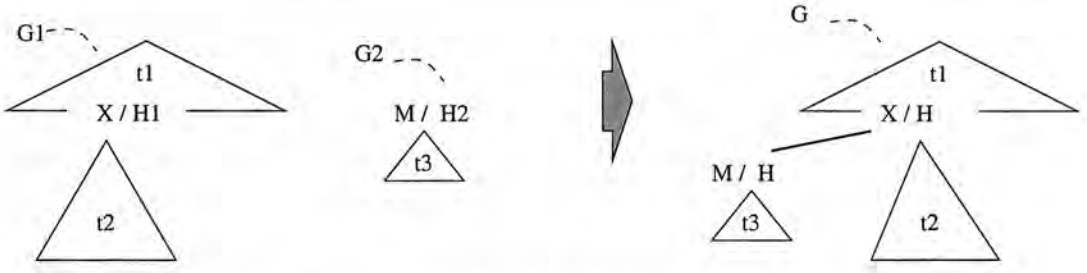


Figure 5.29: Schema two for bottom-up generation

Schema two. Given a structure with an internal nonterminal node annotated with X/H_1 with a corresponding graph G_1 and a structure with a top node (maximal projection) annotated with M/H_2 and a corresponding graph G_2, it is possible to derive a new structure where M is sister-adjoined at X if (i) X's SAC (sister adjunction) constraints allow the sister adjunction of M and (ii) G_1 and G_2 can be joined at concepts H_1 and H_2. The semantics of the corresponding structure, G, is the join of G_1 and G_2 on concepts H_1 and H_2.

Figure 5.29 illustrates this schema. H is the join of the concepts H_1 and H_2.

In schema two if we combine structure $T \in \mathcal{T}_i$ with a modification structure $M \in \mathcal{T}_j$ (where i and j are the minimal such indices) then the resulting structure is in $\mathcal{T}_{max(i,j)+1}$

Termination condition. Among the derived structures there is one with a top syntactic category which unifies with the input syntactic goal and the associated graph with this syntactic structure *BuiltSem* is between *LowerSem* and *UpperSem.*

We can express bottom-up generation in terms of rewriting of sets of legal partial structures (which is what an augmented DTG derivation is) as follows:

Rewriting step corresponding to schema one:

$$\{Descr^0[\mathsf{X}_1\downarrow,\ldots,\mathsf{X}_n\downarrow],\ T_1^{i-1},\ldots,T_n^{i-1},\ \ldots\}\ \Rightarrow$$

$$\{Descr^0 \cup T_k^{i-1} \bullet \{X_k/proj\_node(T_k)\}_{1\leq k\leq n},\ \ldots\}$$

The superscripts are there to indicate from which $\mathcal{T}$ set of legal partial structures the corresponding descriptions come from. In particular we require that $Descr^0$ be from the initial set of chosen mapping rules. As for the complement structures $T_1^{i-1},\ldots,T_n^{i-1}$ we assume that $i-1$ is the smallest index such that all $T_k^{i-1} \in \mathcal{T}_{i-1}(1 \leq k \leq n)$. Note that some T_j^{i-1} could already be present in $\mathcal{T}_l$ where $l \leq i-1$.

Rewriting step corresponding to schema two:

$$\{Descr^i[\mathsf{X}*], ModTrD^j \ldots\} \Rightarrow \{Descr \cup ModTrD \cup idom(\mathsf{X}*, root(ModTrD)) \ \ldots\}$$

It is worth pointing out that the presented bottom-up generation strategy is conceptually very close to shift-reduce parsing for CFGs.

The notion of derivation is, by definition, a declarative one. It is not specific to generation only but can be used to define what can be valid structures of parsing.

5.10.3 Derivation and generation

The notion of derivation we have defined so far describes the relationship between syntactic structures (trees) and the semantic structures conveyed by them. But in our generation framework we have the input semantics which can be slightly different from what is actually conveyed. And how are the lower and upper semantic constraints used? What is the relationship between built semantics and the input semantics? What does applying a MR mean? We look at these and other questions in this section. We will try to relate in a declarative fashion the inputs of the generator to its outputs. This section contains a generic characterisation of generation—not an algorithm.

For our purposes we will need to define the notion of the combination of the input semantics with the semantic contribution of MRs which we call the polluted input semantics (*Poll_In*). This semantic structure is 'polluted' because MRs can introduce

additional semantic material and because we interpret maximal join as a destructive operation[32].

Definition 5.17 (Polluted input semantics)

The polluted input semantics is the result of the maximal join of InputSem with the applicability semantics of the MRs *that have been successfully applied up to a certain point in the derivation process.*

The *BuiltSem* is built incrementally as the generation progresses. Initially it is the empty conceptual graph. Because of what *Poll_In* is, it is always the case that:

$$Poll\_In \leq BuiltSem \tag{5.7}$$

Because *InputSem* and *BuiltSem* are generalisations of *LowerSem* (equations 5.1 and 5.2) and the fact that *Poll_In* can be seen as the maximal join of *InputSem* and *BuiltSem* it follows that:

$$LowerSem \leq Poll\_In \tag{5.8}$$

Now we turn to what we mean by applying a MR from a semantic point of view. Figure 5.30 complements the textual description. Suppose we have a MR in the database which we call *Generic Mapping Rule*. The generator will need to create a copy or an instance of the generic mapping rule. This is similar to what happens in languages like PROLOG or LISP where at execution time it is copies of the actual definitions of clauses or functions that are manipulated. This way the same MR can be used over and over again in a derivation by using different instances of it. This is step one indicated in Figure 5.30. The test of whether the MR can be applied is maximally joining the copy of the MR's applicability semantics (AS) with *Poll_In* (the initial value of *Poll_In* is a copy of *InputSem*). In this process we, in general, will pollute both structures. We do want information to propagate to AS because we want the internal generation goals to be instantiated before exploring them. On the other hand we want to keep AS unpolluted so that we can register the contribution of the MR towards the meaning

[32] In this respect we are close to the way unification is traditionally viewed in computational linguistics, yet differ from all other implementations of maximal join for CGs in which the non-destructive interpretation is used.

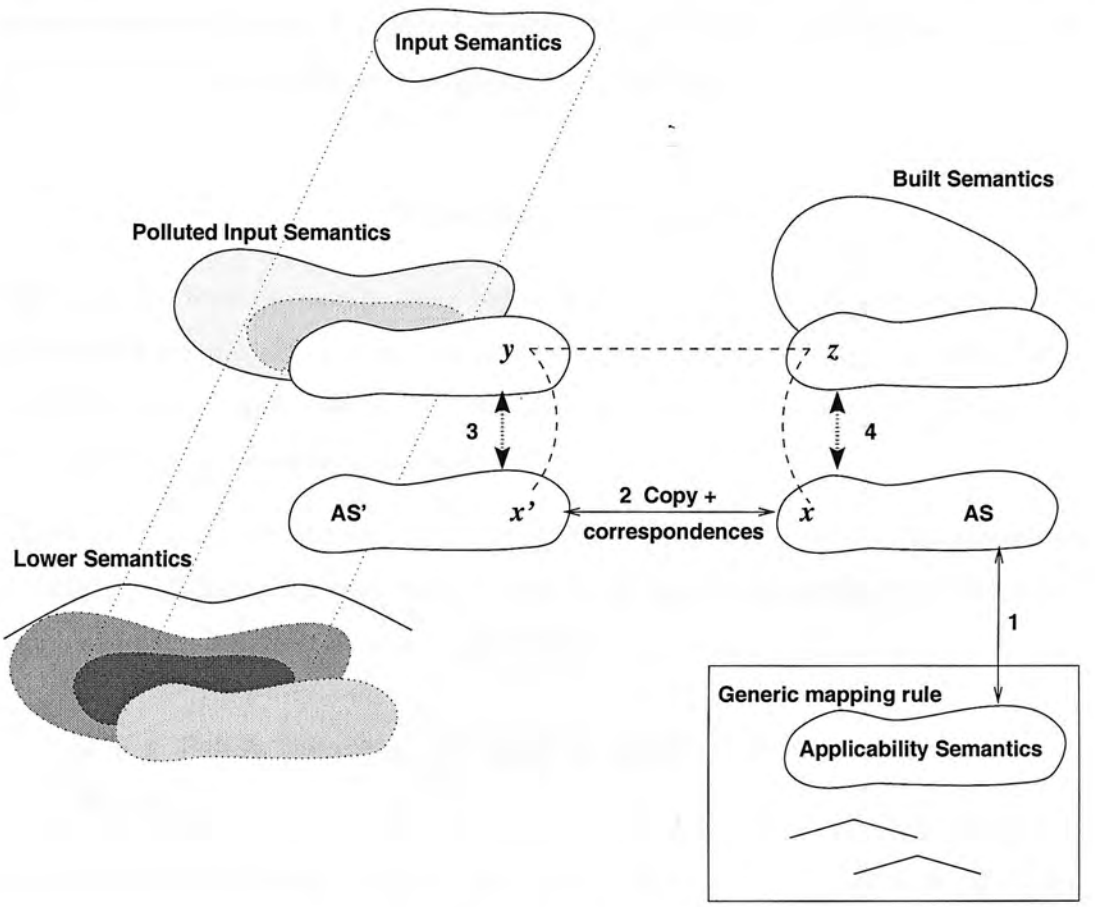


Figure 5.30: Applying mapping rules

of the generated phrase (*BuiltSem*). In order to handle this an extra copy of the applicability semantics is created AS' and the correspondences between the elements of both applicability semantic structures are established. This correspondence will be needed in determining how to combine AS with *BuiltSem*. This is step two indicated in Figure 5.30. Step three consists of maximally joining AS' with *Poll_In*. Step four is deciding how AS is going to be combined with *BuiltSem*. First of all let us note that because of the nature of *Poll_In* (namely that it is the maximal join of *InputSem* and *BuiltSem*) every element X (concept or relation) in *BuiltSem* must have a counterpart in *Poll_In*. Formally we assume we have a total function:

$$\mathcal{B}(X_{BuiltSem}) = X_{Poll\_In} \tag{5.9}$$

We then also have the isomorphism between AS and AS'—$\mathcal{J}$. In step three we also established the projection π between the graph AS' and its counterpart after AS' is maximally joined with *Poll_In*. Now we are in a position to specify how an element x from AS is to be intergrated in *BuiltSem*.

Given an $x \in AS$ we can identify an $x' \in AS'$ such that $x \overset{\mathcal{J}}{\longleftrightarrow} x'$. The projection π uniquely identifies a y such that $y = \pi(x')$. We define the image of x after AS is combined with *BuiltSem* to be a z such that:

1. **if** $y \notin$ *Poll_In* **then** $x = z$ is added as additional element to *BuiltSem* with $\mathcal{B}(z) = y$.
2. **if** $y \in$ *Poll_In* **and** $\mathcal{B}^{-1}(y)$ is not defined, **then** $x = z$ is added as additional element to *BuiltSem* with $\mathcal{B}(z) = y$.
3. **if** $y \in$ *Poll_In* **and** $\mathcal{B}^{-1}(y)$ is defined, **and** $\mathcal{B}^{-1}(y) = z$ **then** x is joined with z.

Please refer to Figure 5.30. This method of relating x to its image z after AS is combined with *BuiltSem* defines a valid maximal join between AS and *BuiltSem* because for all X' joined with images of elements from *BuiltSem* the corresponding x'es are joined with the original elements in *BuiltSem*. Steps three and four can also be thought of as being performed in parallel.

The sequence of operations on *AS* and *BuiltSem* corresponds to a derivation in terms of Section 5.10. Also note that *BuiltSem* can be a disconnected graph—nothing in the discussion above relied on *BuiltSem* being a connected graph! This is important for non-blackboard generation approaches like bottom-up generation.

5.10.3.1 Checking boundary constraints

We now look at the role of the boundary semantic constraints. We assume the existence of two functions $\mathcal{L}$ and $\mathcal{U}$ defined as follows. $\mathcal{L}$ maps every concept C_{Poll_In} in *Poll_In* onto its corresponding concept in *LowerSem*, $C_{LowerSem}$:

$$\mathcal{L}(C_{Poll\_In}) = C_{LowerSem} \tag{5.10}$$

$\mathcal{U}$ maps every concept C_{Poll_In} in *Poll_In* onto its corresponding set of concepts in *UpperSem* or, if the set is empty, $\mathcal{U}$ yields $\{\boxed{\top}\}$:

$$\mathcal{U}(C_{Poll\_In}) = \begin{cases} C_{UpperSem} & \text{if such exists} \\ \boxed{\top} & \text{otherwise} \end{cases} \tag{5.11}$$

We can think of $\mathcal{L}$ and $\mathcal{U}$ as being given with the input to the generator. $\mathcal{U}$ never needs to change while $\mathcal{L}$ is updated when the applicability semantics of a MR adds additional semantics to *Poll_In*.

Finally, we need to relate some elements from *Poll_In* to their originals in *InputSem*. We will assume we have a partial function $\mathcal{I}$:

$$C_{InputSem} = \mathcal{I}(C_{Poll\_In}) \tag{5.12}$$

Using $\mathcal{B}$, $\mathcal{L}$ and $\mathcal{U}$ it is possible to relate every element from *BuiltSem* to its counterpart elements in *LowerSem*, *InputSem* and *UpperSem*:

$$\begin{aligned} C_{LowerSem} &= \mathcal{L}(\mathcal{B}(C_{BuiltSem})) \\ C_{InputSem} &= \mathcal{I}(\mathcal{B}(C_{BuiltSem})) \\ C_{UpperSem} &= \mathcal{U}(\mathcal{B}(C_{BuiltSem})) \end{aligned} \tag{5.13}$$

We can now go back to constraint 5.4:

$$C_{LowerSem} \leq C_{BuiltSem} \leq C_{InputSem} or C_{InputSem} \leq C_{BuiltSem} \leq C_{UpperSem}$$

At the end of the derivation this constraint has to hold. As far as processing is concerned this constraint can be checked at the end of the derivation. However, it might happen that it is possible to determine that the constraint will fail much earlier in which case the generator need not perform useless actions the results of which will have to be abandoned at the end. The first thing to note in this respect is that $C_{BuiltSem}$ always becomes more specialised during processing. Now let's consider the case when a copy of the applicability semantics of a MR AS has been maximally joined with *BuiltSem*.

1. **if** a concept from *BuiltSem* $C_{BuiltSem}$ becomes overconstrained and $C_{LowerSem} \leq C_{BuiltSem}$ no longer holds, **then** the generator should fail immediately.

2. **if** a concept from *BuiltSem* $C_{BuiltSem}$ becomes more specific than $C_{InputSem}$ **then** constraint $C_{BuiltSem} \leq C_{InputSem}$ no longer needs to be checked (because it will always be true).

Nothing much can be said about $C_{InputSem} \leq C_{BuiltSem} \leq C_{UpperSem}$ half way through the processing except perhaps that:

if $C_{UpperSem} \leq C_{BuiltSem}$ **then** $C_{BuiltSem}$ had better be referred to in one of the generation goals because otherwise it might happen that nothing will further instantiate it later. Alas, even if it is referred to in a generation goal, there is no guarantee that it will be later instantiated enough so that $C_{BuiltSem} \leq C_{UpperSem}$ will hold. So in general we cannot check this constraint until the end.

5.11 Implementation

We have developed a sentence generator called PROTECTOR (approximate **PRO**duction of **TE**xts from **C**onceptual graphs in a declara**T**ive Framew**OR**k). Historically we started with a PROLOG implementation which used Tree-Adjoining Grammar (TAG) [Nicolov *et al.* 95]. Our early work was based on ideas of head-driven parsing and

Gertjan van Noord's implementation of TAGs [vanNoord 93, page 136]. A new programming language LIFE [Aït-Kaci & Podelski 93] (Logic Inheritance Functions and Equations) appeared shortly which had features making it easier for the linguist to describe the mapping rules. D-Tree Grammars formalism was published in ACL'95 [Rambow *et al.* 95a] and we quickly appreciated their advantages for generation. DTG's closeness to TAG meant we had to simplify the TAG system. As the development of LIFE subsided after DEC closed its research labs, and in particular the lack of a compiler for LIFE, we revised our earlier PROLOG generator.

5.11.1 Feature structures encoding

The nodes in the d-trees are labelled by feature structures. So are the preterminal nodes in the lexicon. For efficient processing we use automatic *term encoding* of the feature structures [Mellish 92] which is transparent for the end user. The idea is to transform the feature structures first into terms and then do the unification on the terms. Unification of terms is much more efficient in PROLOG. When we need the feature structures they can be recovered from the term by appropriate *decoding*. Figure 5.31 (from [Mellish 92, p.192]) gives a commutative diagram exemplifying the idea.

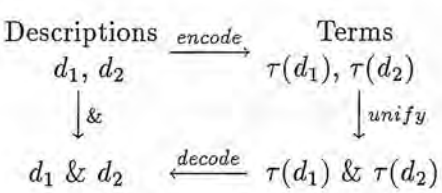


Figure 5.31: Feature structure encoding

In our case the user writes descriptions like `np([case:nom,per:3,num:sg])` which are converted into `np(sg,3,nom,_,_)`. Such an encoding is possible because the feature structures have a finite number of attribute paths and can be converted to terms with fixed arity. The system needs to know the geometry of the feature structure for a given category. Consider the case of an NP (Figure 5.32):

Strictly speaking we only need the names and embedding of attributes. The range of attributes is useful for further improvement using bit-vector encodings [Aït-Kaci *et al.* 89].

$$\text{NP}\left[\begin{array}{ll} \text{AGR:} & \left[\begin{array}{ll} \text{NUM:} & \{\text{SG, PL}\} \\ \text{PER:} & \{1, 2, 3\} \end{array}\right] \\ \text{CASE:} & \{\text{NOMINATIVE, ACCUSATIVE}\} \\ \text{EXPL:} & \{\text{IT, THERE}\} \\ \text{PRO:} & \{+, -\} \end{array}\right]$$

Figure 5.32: Geometry of a noun phrase

From the information about the feature structures the system automatically compiles a feature table for for each category (Figure 5.33).

```
% np(NUM,PER,CASE,EXPL,PRO)

np(_) <-> np(_,_,_,_,_).                                    % 1

np(NUM,_,_,_,_)          :: agr/num      :NUM.              % 2
np(_,PER,_,_,_)          :: agr/per      :PER.              % 3
np(_,_,CASE,_,_)         :: case         :CASE.             % 4
np(_,_,_,EXPL,_)         :: expl         :EXPL.             % 5
np(_,_,_,_,PRO)          :: pro          :PRO.              % 6

np(NUM,PER,_,_,_)        :: agr          :agr(NUM,PER).     % 7
```

Figure 5.33: Feature table for an NP

In line one the `np(_) <-> np(_,_,_,_,_).` the arity of the NP encoding is fixed. The left hand side structure matches what the user uses `np([case:nom,per:3,num:sg])`. Line 2–6 relate a complete attribute path to an argument position in the encoding. The flat encoding might suggest that we are loosing the internal structure of the original feature structure. While not present in the term encoding this information is retrievable. Otherwise the decoding stage in the commutative diagram would not hold. It is also possible to get the value of a non-atomic attribute path. This is shown in line 7. Subject-verb agreement will refer to the agreement feature (as expected) and not individually to all leaf features. These innocent descriptions are actual PROLOG clauses and can be executed to either (1) instantiate a feature or (2) find its value.

The assignment of positions to attribute paths is decided empirically on the basis of a grammar and a test corpus of example semantic structures that are generated. As the system proceeds failures of unification are investigated and *all* clashing atribute paths are recorded. Note this is a "slow" mode of operation as normally unification checks the arguments in their consecutive order and the unification fails as soon as one clashing pair of arguments are found. In doing this additional booking we find attribute paths with high counters of clashing. These are paths with high discriminating power and they are assigned an earlier argument position in the encoding.

For an empirical evaluation of the technique we have used as well as comparison to alternative methods see [Schöter 93].

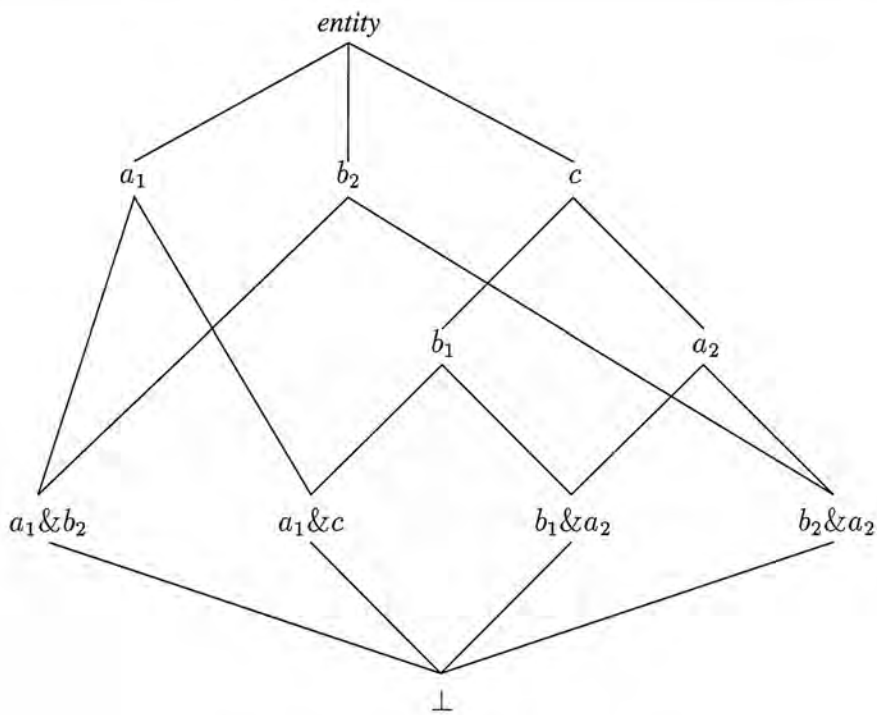


Figure 5.34: Initial type hierarchy

5.11.2 Term encoding of the conceptual hierarchy

We also use term encodings for types from the type hierarchy. Term unification calculates the join of two types. Our approach is based on encoding scheme of Chris Mellish [Mellish 88].

We illustrate the technique by showing an example type hierarchy (Figure 5.34) and the term space that it induces (Figure 5.35). These are the original examples used in the description of the technique [Mellish 88, p.49].

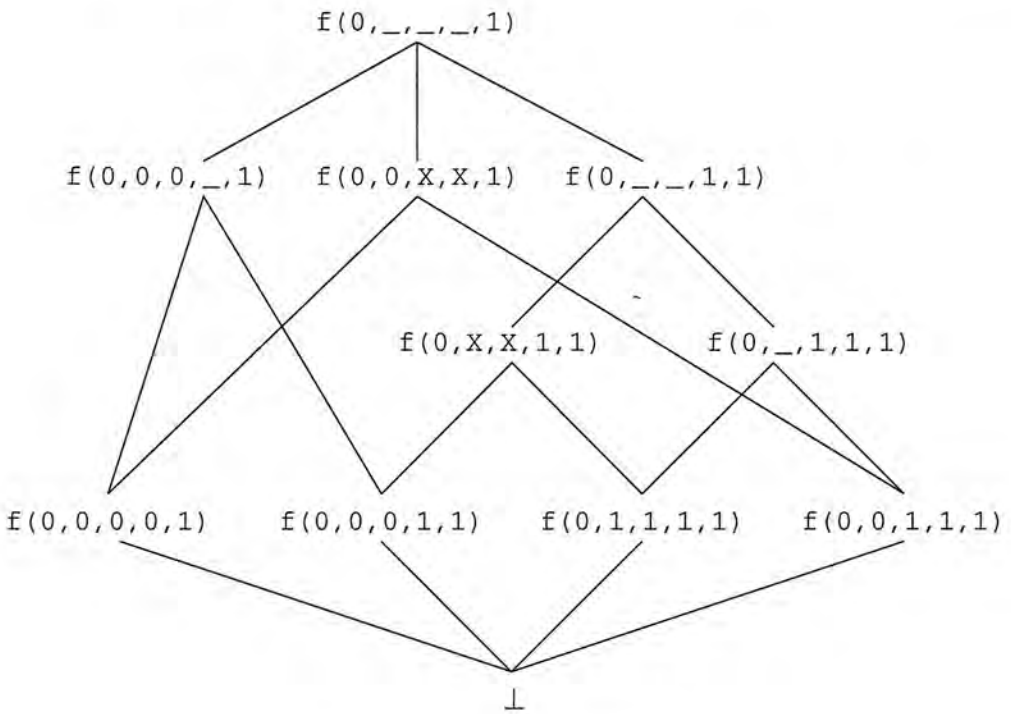


Figure 5.35: Type encodings

Incompatible types (e.g., a_1 and a_2) are encoded as terms which do not unify (`f(0,0,0,_,1)` and `f(0,_,1,1,1)`). The encodings of compatible types (e.g., a_1 and b_2) do unify (`f(0,0,0,_,1)` and `f(0,0,X,X,1)`) and the result is the encoding of their greatest lower bound ($a_1 \& b_2$ with encoding `f(0,0,0,0,1)`). Note the use of shared variables in the terms.

5.11.3 Mapping rules

The linguistic knowledge in PROTECTOR is represented in the mapping rules (grammar) and semantic lexicon. Mapping rules state the relationship between semantics and syntax. Mapping rules are tuples $\langle\ Type, Name, Ex, ASem, DT, Goals, Consumed\ \rangle$ where:

Type is the type of the mapping rule: whether the rule is a modification rule or not and if yes whether it is a pre- or post-modifier.

Name is a name (unique identifier) of the mapping rule used to refer to it (e.g., for debugging purposes).

Ex gives an example of the constructions described by the mapping rule.

ASem is the applicability semantics; a match with it would licence the application of the mapping rule.

DT is a d-tree giving the syntactic part of the construction; nodes in the tree are marked with their corresponding semantic handles.

Goals is a list of internal generation goals.

Consumed is part of the applicability semantics which is consumed by the rule.

```
%--- ACTIVE INTRANSITIVE
%
%         S        [ANIMATE: *a]<-(AGNT)-[ACT: *h]
%       /   \
%     NP     VP
%             |
%             V
%
mr(nonmod,                                  % type
   intransitive,                            % name
   'John runs',                             % example
   H:[act1(H,_),entity(A,_),agnt(H,A)],     % applicability semantics
   dt(  [s +> np, s +> vp, vp +> v],        % immediate dominance
        [],                                 % dominance
        [np < vp],                          % linear precedence
        [s :s([fin:yes])          :H,       % feature labels
         np:np([case:nom,agr:AGR]):A,       % .. and semantic handles
         vp:vp([fin:yes])         :H,
         v :v([vform:fin,agr:AGR]):H ],
        [1],                                % maximal projections: S
        []),                                % equal nodes
   [4,2],                                   % internal generation goals: V, NP
   [agnt(H,A)]).                            % consumed semantics
```

Figure 5.36: Mapping rule in PROLOG

Figure 5.36 shows the intransitive mapping rule as it is written in the linguistic knowledge base. Regarding the d-trees recall from Figure 4.30 (see page 107) the repesentations for d-trees that we used. Two differences are aparent: (1) the mapping rules do

not mention the index (`Idx`) of the d-tree but (2) in the feature labelling the mapping rules also refer to the handles of concepts in the applicability semantics. The index is automatically introduced in a compilation step. Relating syntactic nodes to parts of the applicabilty semantics is the very nature of mapping rules. The maximal projections (a field in the d-tree representation) and the internal generation goals are lists with numbers which refer to elements in the labelling list.

Mapping rules compiler. Mapping rules are initially preprocessed into an internal representation. During this compilation:

1. The concept types in the applicability semantics is term encoded.
2. The nodes in the d-tree are given the same variable index as an argument.
3. The feature structure labels of nodes in the d-tree are also term encoded.
4. The elements of the maximal projection and generation goals lists are replaced by their counterparts in the feature labels list.

Mapping rules can have associated conditions which are often of the kind of checks on the referent fields of concepts (e.g., a mapping rule for a proper name will be used if the referent of the concept is a proper name). In the compilation these conditions are put in the body of the Horn clause that is generated. The compilation is transparent to the user.

5.11.4 Morphological generator

PROTECTOR implements English morphology in a module which takes a feature structure with a preterminal category and information about the morphological stem and returns the inflected form.

$$V \left[\begin{array}{ll} \text{AGR:} & \left[\begin{array}{ll} \text{NUM:} & \textsf{sg} \\ \text{PER:} & 3 \end{array}\right] \\ \text{LEX:} & \textsf{study} \end{array}\right] \Rightarrow \textit{studies}$$

5.11.5 System tracer

The internal representations that PROTECTOR uses differ from what the user enters in the linguistic knowledge base (grammar and lexicon). Thus, in cases when the system does not come up with the desired derivation, it is difficult to use PROLOG's internal tracing to step through the generation process.

We have developed a specialised tracer which pretty prints the internal representations in a form familiar to the user. The tracer, of course, concentrates on the generation aspects of the process and does not go into details of PROLOG's inner workings. The tracer can be switched off. During running time there is a small penalty at each trace point to check whether it is on or not. We will see example of the out of the system tracer in the next section. PROTECTOR keeps a record of the interactions with the system (in a file `protector.log`) in case these need to be examined.

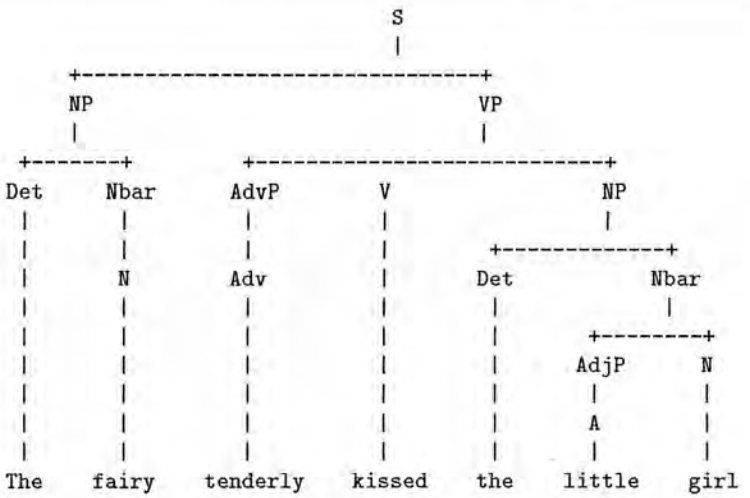


Figure 5.37: Trees in text form

5.11.6 Output formats

PROTECTOR has a graphical interface. We have already seen examples of the graphical displays of trees that it produces in Figure 5.8 on page 129.

For debugging purposes trees are often displayed in ASCII form (see Figure 5.37).

PROTECTOR can export trees in two kinds of LaTeX notation using the packages `qtree.sty`

and `ecltree.sty`. The latter can combine dislay of feature structures using the `avm.sty` package. An example of the output of the `qtree.sty` package is shown in Figure 5.38.

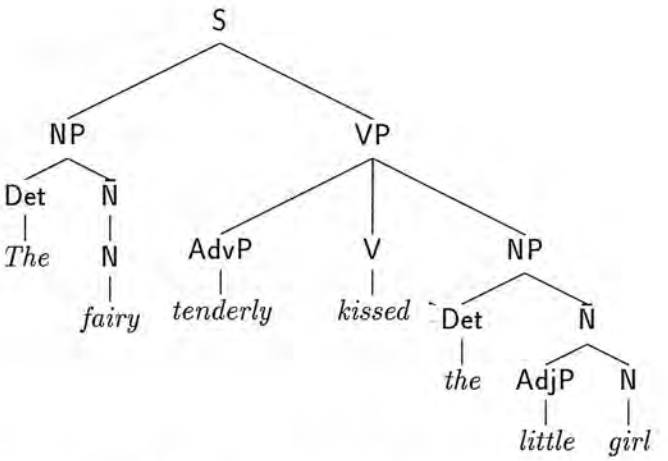


Figure 5.38: Tree output with `qtree.sty`

The display facilities of PROTECTOR have been used as part of the D-Tree Grammar development environment and parsers in the LEXSYS project at the university of Sussex [Carroll *et al.* 98].

We are incorporating converters allowing exporting in SGML format and in schemes for annotation for speech generation [Black & Taylor 97]. In fact speech components require strictly less constituent information than we produce.

5.12 Linguistic coverage

The syntactic coverage of the generator is influenced by the XTAG system (the first version of PROTECTOR in fact used TAGs [Nicolov *et al.* 95]). By using DTGs we can use most of the analysis of XTAG while the generation algorithm is simpler. We are in a position to express subparts of the input semantics as different syntactic categories as appropriate for the current generation goal (e.g., VPs and nominalisations). The syntactic coverage of PROTECTOR includes:

1. intransitive verbs: *John runs*

2. transitive verbs: *The system generated the text*
3. ditransitive verbs: *John gave flowers to Mary*
4. idioms: *John kicked the bucket*
5. topicalisation: *Mary John likes*
6. verb particles: *Mary calmed down*
7. passive: *The text was generated by the system*
8. sentential complements: *John thinks Peter swims well*
9. control constructions: *John wants to run away*
10. relative clauses: *the professor that John admires*
11. nominalisations: *the destruction of the city by the barbarians.*

Adding more mapping rules to cover desired constructions is straight-forward. When the number of mapping rules becomes large, maintaing the linguistic knowledge base will be an issue. We are considering inheritance techniques to capture generalisations as used in the linguistic knowledge based on classification approaches.

5.13 Discussion

The hierarchy of conceptual relations is also a multiple inheritance hierarchy. Presently we use a set of conceptual relations which is comparable to other generation systems (Somer's grid, FUF, VERBMOBIL, KPML). There are proposals for organising conceptual relations in hierarchies both in linguistics (Dowty's entailments of roles [Dowty 91]) and artificial intelligence (Knott's methodology for hierarchical organisation of coherence relations [Knott 96]) and we are in a position to benefit from such developments in a most direct way.

Our generation system is in a sense domain independent in that the generation mechanisms do not depend on the particular ontology being used and if in different application domains it is better to use different ontologies that is possible. Of course the ontology

is reflected in the grammar in the form of the semantic parts of mapping rules and a change in the ontology used for the input semantics needs to be reflected there too. Yet, as the syntactic coverage and the semantic ontology are continually growing in most applications it might be better to consider a mapping of the output structure of, for example a reasoning component, to the input structures of PROTECTOR. Such an approach is pursued in the GIST generation system [Consortium 96] where three different surface generators for three different languages are being used.

In principle it is possible to have only an upper and a lower semantic bound. Yet, in such a model it is difficult to state that a certain type is preferred unless both the upper and lower bound are made equal to this type. However, this severely restricts the flexibility of the system. In particular, if there is no way for the generator to express the input by using a concept at that given level of specificity, the generator will fail. By providing as inputs an upper and a lower bound plus an additional type from the input semantics we are in a position to state that the generator is allowed to deviate from the type of the input semantics within the range specified by the upper and lower bounds but derivations which use concepts whose types are closer to the one in the input semantics will be preferred. We take up the issue of preferring one derivation over another in the Chapter 7. A lexical choice model for nouns that allows deviation from the input has been considered in [Reiter 91]. He is only concerned with choosing individual open class words (nouns) while our model covers uniformly both syntactic and lexical decisions.

During generation it is necessary to find appropriate mapping rules. However, at each stage a number of rules might be applicable and the consequences of choosing a particular mapping rule might be far-reaching and not known in advance. Due to possible interactions between some rules the generator may have to explore different allowable sequences of choices before actually being able to produce a sentence. Thus, generation is in essence a search problem. In order to guide the search a number of heuristics can be used. In [Nogier & Zock 92] the number of matching nodes has been used to rate different matches, which is similar to finding maximal reductions in [Iordanskaja & *et al.* 91]. Considering other aspects like restricting the type and referent labels of concepts can also be exploited. Alternatively a notion of semantic distance

[Foo *et al.* 89] might be employed. In PROTECTOR we use a much more sophisticated notion of what it is for a conceptual graph to match better the initial semantics than another graph. This captures the intuition that the generator should try to express as much as possible from the input while adding as little as possible extra material.

Augmenting the input semantics with information from the applicability semantics of mapping rules is an instance of providing language specific defaults to an incomplete input. Alternative default mechanisms in NLG are discussed in [Harbusch *et al.* 94].

In constructing the rules for mapping semantics (argument structures) to syntax (constituent structures) we have used a notion of thematic ordering or prominence of the conceptual relations (valence roles). The thematic prominence is defined as the order relation:

agent < *beneficiary* < *experiencer/goal* < *instrument* < *patient/theme* < *locative*

The generator attempts to consume the conceptual relations in that order (as a by product of how the mapping rules are organised and not due to a special mechanism). This allows for more syntactic information to be gathered as early as possible so that the derivation process is constrained maximally. Another notion that can guide the choice of initial mapping rules can be the notion of covering more salient aspects of the input structure first. There are some proposals in the literature on visual salience, e.g., [McDonald & Conklin 82] yet having these would mean putting back hierarchical information into our input and we want to keep the two issues separate. Furthermore a good cognitive study of what salience actually means and how it is used is a prerequisite in order for generation systems to be able to benefit from it.

Our approach can be considered incremental to the extent that it might be possible to allow for the input semantics to be augmented during the generation process. By augmenting we mean adding additional information and not overriding previously stated input. We do not look at non-monotonic mechanisms for undoing already generated output which is what is required in order to handle 'change-of-mind' alterations to the input. In scenarios where timeouts are used (e.g., modelling human performance, or simulation of certain disabilities) our approach can be used very successfully because we start looking first for a skeletal structure. If interruptions occur then we are more likely to have a structure that can be converted to a complete tree.

In the Essential Arguments Algorithm Strzalkowski reorders the goals so that more instantiated goals will be explored first (by PROLOG) [Strzalkowski & Martinovic 92]. Surface order is preserved by threading difference lists. Shieber, however, has presented arguments that this cannot be done in general. Due to the extended domain of locality of DTGs we can generate the elements from the internal generation goals in the order in which they will be uttered. Yet, the anchor of the construction might not happen to be the first constituent of the MR tree. Yet, once the anchor is chosen the order in which its arguments are generated is not an issue.

Our generator is not coherent or complete (i.e., it can produce sentences with more general/specific semantics than the input semantics). We try to generate sentences whose semantics is as close as possible to the input in the sense that they introduce little extra material and leave uncovered a small part of the input semantics. We keep track of more structures as the generation proceeds and are in a position to make finer distinctions than was done in previous research. The generator never produces sentences with semantics which is more specific than the lower semantic bound which gives some degree of coherence. Our generation technique provides flexibility to address cases where the entire input cannot be expressed in a single sentence by first generating a "best match" sentence and allowing the remaining semantics to be generated in a follow-up sentence.

We use a notion of headed conceptual graphs, i.e., graphs that have a certain node chosen as the semantic head. The initial semantics need not be marked for its semantic head. This allows the generator to choose an appropriate (for the natural language) perspective. The notion of semantic heads and their connectivity is a way of introducing a hierarchical view on the semantic structure which is dependent on the language. When matching two conceptual graphs we require that their heads be the same. This reduces the search space and speeds up the generation process. Headed conceptual graphs have also been considered in the conceptual graphs literature as means of speeding up the maximal join of two conceptual graphs when it is clear what the top-level concepts are [Fall 95].

Imperfect match between the input semantics and the applicability seantics of mapping rules is considered in the JAPANGLOSS MT system [Knight & Hatzivassiloglou 95]:

"The feature `:rest` is our mechanism for allowing partial matching between rules and semantic inputs. Any input features that are not matched by the selected rule are collected in `:rest` and recursively matched against other grammar rules."

Our adoption of a parallel correspondence architecture is not unlike similar approaches in linguistics—e.g., the LFG theory [Bresnan 82], cognitive science—Jackendoff's theory of the language faculty [Jackendoff 95], and phonology where mapping between parallel structures is the framework of choice for most researchers.

Our approach can be seen as a generalisation of semantic head-driven generation [Shieber *et al.* 90]—we deal with a non-hierarchical input and non-concatenative grammars. The use of lexicalized DTG means that the algorithm in effect looks first for a syntactic head. This aspect is similar to syntax-driven generation [König 94].

Potentially the information in the mapping rules can be used by a natural language understanding system too. However, parsing algorithms for the particular linguistic theory that we employ (DTG) have a complexity of $O(n^{4k+3})$ where n is the number of words in the input string and k is the total number of d-edges in the elementary trees [Rambow *et al.* 95b]. This is a serious overhead and we have not tried to use the mapping rules in reverse for the task of understanding.[33] In the grammar fragment that we are using we do not have more than 2 d-edges in elementary structures.

The algorithm has to be checked against more linguistic data and we intend to do more work on additional control mechanisms and also using alternative generation strategies using knowledge sources free from control information.

5.14 Conclusions

We have presented a technique for sentence generation from conceptual graphs. The use of a non-hierarchical representation for the semantics and approximate semantic matching increases the paraphrasing power of the generator (i.e., more sentences can be produced) and enables the production of sentences with radically different syntactic structure due to alternative ways of grouping concepts into words. In generation

[33] At present we are working on robust parsing and we are pursuing a head-driven approach.

from hierarchically structured representations this can be done only by performing transformations on the input semantics because the input already contains certain language commitments. This is particularly useful for multilingual generation and in practical generators which are given input from non-linguistic applications. The use of a syntactic theory (D-Tree Grammars) allows for the production of linguistically motivated syntactic structures which will pay off in terms of better coverage of the language and overall maintainability of the generator. The syntactic theory also affects the processing—we have augmented the syntactic operations to account for the integration of the semantics. As a result of the way the semantics is 'consumed', our generator is not constrained to produce sentences with semantics either more specific or more general than the input semantics. We have deliberately aimed at a generation architecture which guarantees that the sentence's semantics covers as much as possible from the input semantics and leaves out as little as possible. The generation architecture makes explicit the decisions that have to be taken and allows for experiments with different generation strategies using the same declarative knowledge sources.

Increased paraphrasing power means that generators can produce more alternatives/ paraphrases for a given semantic input. This has two important ramifications:

1. A generator with higher paraphrasing power is more likely be able to express successfully a given semantics than a rigid generator.

2. Such a generator can then choose among a bigger range of constructions one (or a number) which satisfies certain (non-semantic) conditions. Of course the choices the generator makes during generating a sentence and those which constrain the form of the output (stylistic, pragmatic choices) can be interleaved.

SUMMARY

- ⊙ Mapping rules state how semantic information can be expressed linguistically.
- ⊙ Mapping rules are declarative and can be used with different strategies.
- ⊙ Derivations are described for generation too.

- ⊙ Approximate semantic matching allows the generator to express less/more information. Paraphrases need not have the same semantics as the input. PROTECTOR uses a more general notion of completeness and coherence.
- ⊙ Generation can be viewed as a rewriting system.
- ⊙ Lexical choice and syntactic choice is interleaved in PROTECTOR.
- ⊙ Top-down generation is a goal-driven process.

The generator works well but when a certain path of the search space is to be abandoned or more paraphrases are needed (i.e., when backtracking occurs) some work is duplicated. In the next chapter we will look at ways to explore memoing techniques which avoid this duplication.

Chapter 6

Chart-Based Generation

"As I knew, or thought I knew, what was right and wrong,
I did not see why I might not always
do the one and avoid the other.
But I soon found I had undertaken
a task of more difficulty that I had imagined."
—Benjamin Franklin

In this thesis we have considered a more general view of generation which leads to an even worse search space than usual. Hence it is especially important to address search control. This chapter goes deeper in complicating the generation model—we look at the use of memoization techniques for generation. One of the best currently known techniques for parsing which uses memoization is chart parsing. We briefly survey this technique and show that a chart can also be used for generation. The idea of using a chart in generation has been suggested in previous research and we look at the existing proposals. These proposals use an indexing on the string positions just as in parsing. However, using the same mechanisms for generation does not offer substantial advantages because in generation the string positions of individual words are not known in advance. We define a new notion of chart generation which uses indexing on the semantic structures.[1] The new formulation is first illustrated for grammars using a

[1] We developed the memoization technique following a discussion with Martin Kay in July 1995 at ELSNET's summer school when he made the observation that we will be duplicating work on backtracking. Kay also told us about the approach by Haruno et al. which attempts to fix the indexing on string positions (we review this approach in Section 6.2.2). Consequently Kay published a paper on Chart Generation which is close to our approach yet the use of CFG and the assumption of using complete structures leads to gross inefficiency in the case of modifiers. We discuss this is

context-free backbone and then for non-concatenative grammars. PROTECTOR implements a top-down chart strategy. Our formulation provides a very easy way to define an array of alternative processing strategies. To this end we introduce agenda-based control for chart generators and different generation strategies can be seen as different ways of exploring the agenda. This view of looking at generation suggests that some aspects of the generation process can be done differently and we discuss the space of chart generators. Having a system that allows for easy definition of different generation strategies provides for the eventual possibility of comparing different algorithms based on the uniform processing mechanism of the agenda-based control for chart generation.

Overview

We start by motivating memoization techniques and the use of charts in Section 6.1. Then in Section 6.2 we develop the model for chart generation: we first introduce some terminology (Section 6.2.1) in the area of tabular methods for parsing as we are going to use very similar techniques; then we review the existing approaches to tabular and chart-like techniques (Section 6.2.2); then we provide a new formulation (Section 6.2.3) initially for grammars using a context-free skeleton (Section 6.2.3.2) which makes the jump smaller to the heart of the chapter—memoization generation techniques for non-concatenative grammars (Section 6.2.4). We discuss specific aspects of handling DTGs in Section 6.2.4.1 and how we can recover the needed syntactic structures in Section 6.2.4.2. We go through a detailed example (Section 6.2.5). Agenda based processing is postponed till the next chapter. We conclude with a discussion of the tabular approach (Section 6.3).

6.1 Motivation: doing the work once

In this section we discuss the use of memoization techniques in generation. The idea of memoization is very simple: *Storing the results of complex computations might speed other computations that use these results later on*[2]. Using memoization techniques is

Section 6.3.2.

[2] The term memoization was coined by Donald Michie to refer to the process by which a function is made to *automatically* remember the results of previous computations. We will use memoization

justified in cases when:

- The results of previous computations are likely to be needed in the future.
- It is 'cheaper' to retrieve the old value than to compute it again.

Memoization is relevant to generation approaches based on backtracking because it may happen that computations in branches of the search space that have been abandoned have to be recomputed. In Section 1.6 we discuss non-deterministic generation models and focus on the causes of failure. At the end of the section we show an example of a lexical gap in which the generator has no way of knowing that it might reach a dead end and thus will need to reconsider previous decisions. We will see that the results of previous computations will be needed later on and that it can be cheaper to retrieve them than compute them from scratch.

6.1.1 Examples of local failures in generation: Lexical gaps

In this section we examine closer the reasons for failures in generation. Our purpose is twofold:

1. to motivate a non-deterministic mode of processing, and
2. to show that even in unsuccessful branches of the generation search space generators do useful work which can be reused instead of being thrown away.

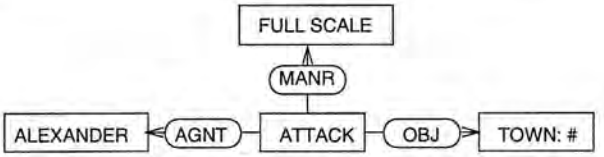


Figure 6.1: *Alexander attacked the town. The attack was fullscale.*

We now introduce lexical gaps through an example. Suppose we have the semantics in Figure 6.1 which we want to generate as a sentence (S). This structure is interesting because at the onset of generation there are at least two mapping rules that match the input (see Figure 6.2) each corresponding to the sentences:

slightly more generally.

(6.1) *Alexander attacked the town 'full-scalely'.*
Alexander launched a full-scale attack on the town.

Choosing the first mapping rule in Figure 6.2 and continuing the generation process from there leads to failure—we cannot successfully express the concept FULL SCALE as an adverb (ADV).[3] Yet, the generator does not know that in advance so it 'falls in the trap'. Note that even under a very fine grained notion of what input structures look like (i.e., one where topicalised structures would be distinguished) the input structures of the declarative sentence and the light verb counterpart are very unlikely to be different.

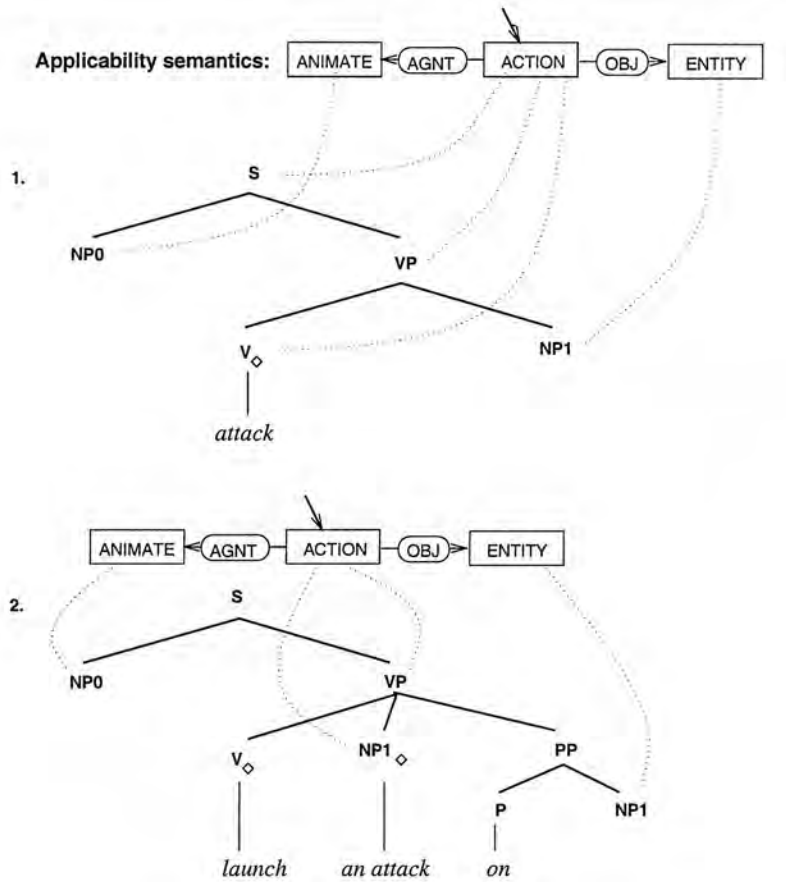


Figure 6.2: Top level mapping rules

Let us examine closer the search space that the generator explores (see Figure 6.4). The generator choose the transitive mapping rule and generates the skeletal structure *Alexander attacked the town* by following the internal generation goals of the initial

[3] Here we assume that we want to generate all of the input semantics.

mapping (we have marked the frontier nonterminal (goal) nodes in bold). Then in phase two the generator attempts to consume the remaining semantics. This is done in the following way: a mapping rule is chosen (non-deterministically) which consumes some of the remaining semantics and the generator tries to integrate the structure of the mapping rule in the current (global) structure that it builds.[4] Generation will terminate successfully if we can consume all the input. In our case the generation will fail because there is no mapping rule (not only in the linguistic knowledge base of the generator but worse of all in the English language) that would allow us to express the remaining semantics as a structure that we can intergrate to the current global structure. We want to express the concept [FULL-SCALE] but cannot.

Here is an example from [Zock 87]:

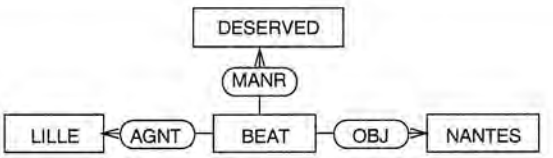


Figure 6.3: *Lille defeated Nantes. The victory was (well) deserved*

(6.2) French: * *Lille a battu Nantes* ***méritemment.***

(6.3) German: *Lille hat Nantes* ***verdient*** *geschlagen.*

`NP + Aux + NP + Adverb + Verb`

The concept [DESERVED] cannot be expressed as an Adv in French but it is possible to do so in German.[5] In English similarly there is a lexical gap:

(6.6) ? *Lille (* well) deservedly defeated Nantes.*

(6.7) *Lille achieved a (well) deserved victory over Nantes.*

[4] More details about the generation model are given in the previous chapter (5).

[5] In French one can incorporate the information in a NP:

(6.4) *le mérite de la victoire de Lille sur Nantes ...*

(6.5) *la victoire méritée de Lille sur Nantes ...*

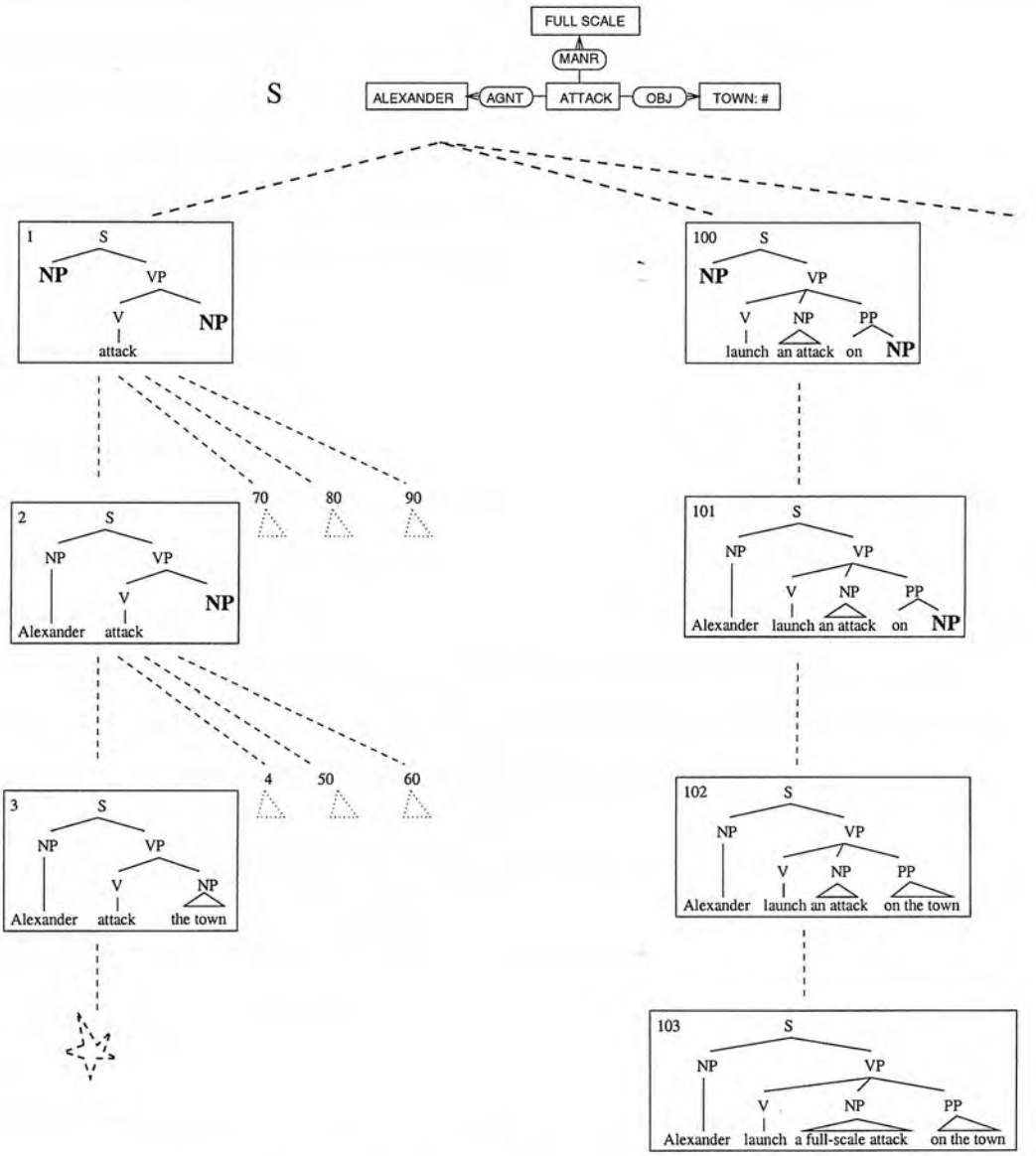


Figure 6.4: The search space for the example

Returning to the search space of example 6.1 we observe that the current global structure before the failure was detected (numbered 3 where the numbers correspond to the chronological order in which the search space is explored) contains sub-structures that appear in the final result (structure 103). In particular two of the NPs in that structure are exactly the same as those in the end (103). Unfortunately backtracking algorithms when they reach a dead-end 'undo' all computations until the previous choice point. In our example the generator would have to undo (forget) about all the structures it had built all the way up to the point when it chose the wrong mapping rule. This was the first choice that was made so practically every computation is lost. All the work that went into building the two NPs has to be duplicated.

6.1.2 Memoization

In order to avoid redoing computations we consider a kind of memoization technique. The generator keeps the results of previous computations and whenever the generator needs to redo the same computations it simply looks up the results. In the lexical gap example the two NPs that were generated in the skeletal structure of the first mapping rule appear in the final structure (103). Had we somehow kept them we could use them when we needed them again for the final result. In order to make these ideas more concrete we need to address the following questions:

1. What do computations and their results look like?
2. How are results of previous computations kept?
3. When are two computations the same?

We look into these in the next section (6.2) where we consider one particular kind of memoization based on a data structure — a chart — which is used as the memory of previous computations.

6.2 Chart generation from non-hierarchical structures

This section provides a new formulation of chart-based generation for systems that take non-hierarchically structured semantic representations. We formulate chart generation from a semantic point of view and look at the use of declaratively stated mapping rules relating the semantic and the syntactic structures. We consider context free grammars as well as non-concatenative grammars in the family of Tree-Adjoining Grammars (Lexicalised D-Tree Grammars).

We first briefly summarise the idea of chart parsing in Section 6.2.1 in order to introduce terminology which we will use extensively later. A second reason for us to look at chart *parsing* is because the initial attempts to use charts for generation were very similar to the parsing techniques. We then survey existing proposals for using charts in generation (Section 6.2). In Section 6.2.3 we formulate chart generation for context-free grammars whose nodes are annotated with semantic information represented in a non-hierarchical formalism and we give an example of generating a sentence in Section 6.2.3.2. In Section 6.2.4 we offer the details for chart generation using DTGs.

6.2.1 The notion of a chart

Active charts are a data structure which is at the heart of a generalisation of a parsing algorithm for context free grammars initially proposed by Jay Earley in the early 1970s [Earley 70]. Chart parsing was introduced in Computational Linguistics by Martin Kay [Kay 80].[6] The details of this section are based on an earlier version of [Ritchie 97].

> Abstractly a chart is a data structure which contains information about partial and completed analyses of substrings of the input sentence. Substrings of the input sentence are identified by position numbers.[7]

---

[6] Introductions to chart parsing can be found in [Thompson & Ritchie 84, Gazdar & Mellish 89, Covington 94].

[7] Here is an example of a sentence with the string positions:

	She	likes	Paris	
	↑	↑	↑	↑
	0	1	2	3

Definition 6.18 (Dotted rule)

Given a CFG G of the form $\langle V_T, V_N, S, P\rangle$, a dotted rule based on G is $r \rightarrow l_1 \bullet l_2$ where $r \rightarrow l \in P$ and l_1 and l_2 are substrings of l such that $l_1 l_2 = l$. Where either l_1 or l_2 is empty, they are omitted from the expression.

Definition 6.19 (Chart)

Given a CFG G of the form $\langle V_T, V_N, S, P\rangle$, and a sequence of terminal symbols $a_1 a_2 \ldots a_n \in V_T^*$, a chart based on $a_1 a_2 \ldots a_n$ and using G is a set C of triples $\langle i, j, r\rangle$ which meets the following conditions:

1. *for every $\langle i, j, r\rangle \in C$ it holds that $0 \leq i \leq j \leq n$ and r is a dotted rule based on G.*
2. *for every $a_i \in \{a_1, a_2, \ldots, a_n\}$, and for every rule of the form $L \rightarrow a_i \in P$, there is an element $\langle i-1, i, L \rightarrow a_i \bullet \rangle \in C$.*

The final stipulation in this definition corresponds to something that would be treated as the initialisation phase of a procedural construction of a chart for a sentence.

Each element of a chart is referred to as an *edge* (since charts are often depicted as directed graphs); an edge of the form $\langle i, j, c \rightarrow c_1 \ldots c_{i-1} \bullet c_i \ldots c_k\rangle$ is referred to as an *active edge*, and an edge of the form $\langle i, j, c \rightarrow c_1 \ldots c_k \bullet \rangle$ is an *inactive edge*. Informally, edges represent constituents found so far by the parser, or rules which have been introduced to the parsing process. The "dot" in a dotted rule denotes progress through the right hand side of the rule, indicating which required constituents have been found so far, and the first two components of the edge indicate the region of the string concerned. Inactive edges represent complete constituents which have been found by the parser and active edges indicate partial constituents which require further constituents to become whole. An active edge $\langle i, i, c \rightarrow \bullet c_1 \ldots c_k\rangle$ is referred to as an *empty* active edge; it represents a constituent none of which has been found yet, but for which a suitable rule has been introduced into the chart.

The central idea of active chart parsing (sometimes known as the *fundamental rule*[8]) is that wherever there is an adjacent compatible pair of an active and an inactive edge, a further edge should also be in the chart to represent the combined information of these two edges. The notion of "compatibility" required here is that the inactive edge represents a constituent of category c, and the active edge requires a constituent of the type c as the next part of the right hand side of its dotted rule. Also, the right end (second component) of the active edge and the left end (first component) of the inactive edge must coincide.

[8] The term *fundamental rule* is due to Henry Thompson. The fundamental rule is also referred to as the *multiplication* rule.

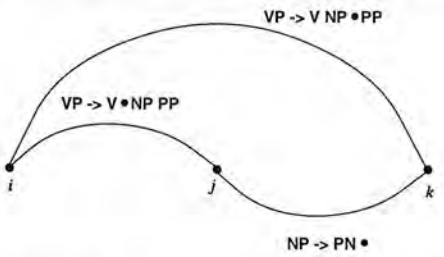


Figure 6.5: Fundamental rule for chart parsing

Definition 6.20 (Fully resolved chart)

A chart $\mathcal{C}$ is fully resolved if for every pair of edges $\langle i, j, c \rightarrow c_1 \ldots c_{m-1} \bullet c_m \ldots c_n \rangle$ (where $m \leq n$) and $\langle j, k, c_m \rightarrow \alpha \bullet \rangle$ (for some $\alpha \in V_N^* \cup V_T$), there is also an edge $\langle i, k, c \rightarrow c_1 \ldots c_m \bullet c_{m+1} \ldots c_n \rangle$ (if $m < n$) or $\langle i, k, c \rightarrow c_1 \ldots c_n \bullet \rangle$ (if $n = m$).

A chart parser is driven by two principles: one is that of edge combination, as given immediately above, and the other is the introduction of rules into the chart. The latter is normally done in one of two ways, either bottom-up or top-down.

Bottom-up rule introduction can be glossed informally as "if there is a complete constituent matching the first category on the RHS of a rule, then an edge for that rule must be present at the start of that constituent":

Definition 6.21 (Bottom-up explored chart)

A chart $\mathcal{C}$ is bottom-up explored if for every inactive edge $\langle i, j, c_1 \rightarrow \alpha \bullet \rangle$ there is also an edge $\langle i, i, c \rightarrow \bullet c_1 \ldots c_n \rangle$ for every rule in the grammar of the form $c \rightarrow c_1 \ldots c_n$.

This should perhaps be known as *left-corner explored.*

Top-down rule introduction can be glossed informally as "if there is an edge seeking a particular category, there must be, at the same position, edges for every rule which could expand that category":

Definition 6.22 (Top-down explored chart)

A chart $\mathcal{C}$ is top-down explored if for every edge of the form $\langle i, j, c \rightarrow c_1 \ldots c_m \bullet c_{m+1} \ldots c_n \rangle$ there is also an edge $\langle j, j, c_{m+1} \rightarrow \bullet \alpha \rangle$ for every rule in the grammar of the form $c_{m+1} \rightarrow \alpha$. Also, there is an (empty active) edge $\langle 0, 0, S \rightarrow \bullet \alpha \rangle$ for every rule of the form $S \rightarrow \alpha$.

6.2.1.1 Chart properties, strategies and algorithms

So far we have talked about properties of charts (e.g., top-down explored). This is perhaps the highest level of abstraction at which we can state what charts are without reference to idiosyncratic details of a particular implementation. Once we know what is it that we want to model (i.e., the properties or behaviour of charts) we can start devising algorithms that meet these criteria. We will be more specific about implementation details regarding generation in Section 6.2.5.

6.2.2 Approaches to chart generation

The active chart technique has also been used for generation [Masahiko & Matsumoto 93, Haruno *et al.* 93]. We present a brief rational reconstruction of the approach.[9] In contrast to parsing in generation we need to associate semantics with its possible realisations and thus the categories of our rules will be more complex. We assume they have the form Syn/Sem where Syn is the syntactic category of the expression and Sem is the semantics. The format of rules becomes:

$$Cat/Sem \rightarrow c_1/S_1, \ldots, c_n/S_n$$

In the following we will represent 'dotted' rules by explicitly labelling the syntactic constituents which have *not* been realised with '?'. The reason for that is that while in chart parsing the dot • separates the syntactic constituents that have been found/expanded (in a left to right fashion!) from those that haven't, in generation it is often preferable to expand certain constituents before others which need not necessarily precede them in the final string.[10] Here a chart will consist not only of edges but also of *forward links*. Links are used in order to express the fact that two edges are adjacent. The semantic head[11] will be marked by **#**.

In the following we first present the rules for a bottom-up chart generation. Then we state the conditions for top-down processing. Active edges are drawn above the

[9] We use the notation we have introduced so far and do not give an example in Japanese.

[10] Often in implementations one orders the constituents in the order in which it would be best to expand them and the linear order of the surface string is determined by other means (e.g., by using the technique of difference lists in Prolog.

[11] As in the SHDG algorithm which we reviewed in Chapter 2 the semantic head is a (complex) category in the LHS of a CFG grammar rule whose semantics is identical to the semantics of the RHS (mother).

vertices using a solid line; inactive (complete) edges are drawn below the edges with a dashed line. Also the edges that are 'added'[12] (i.e., the edges that are on the RHS of the implication in the statement of the schemata) are given in the figures in italics.

Definition 6.23 (Dynamic pivot introduction)

For every edge $\langle i, j, c \rightarrow c_1/S_1 \ldots ?c_k/S_k \ldots c_n/S_n \rangle$ *in the chart* $\mathcal{C}$ *(where* c_k *is the first non-expanded constituent) and all rules of the form:* $\alpha/S \rightarrow$ `word` *such that* S *and* S_k *are unifiable*[13] *there is an inactive edge* $\langle w, w', \alpha/S \rightarrow$ `word`$\rangle$ *and a forward link* $\langle j, w \rangle$ *in* $\mathcal{C}$.

See Figure 6.6.

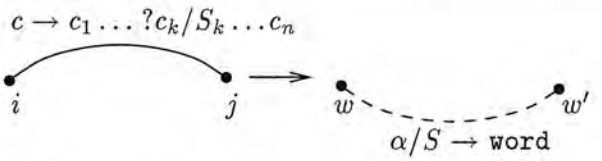


Figure 6.6: Dynamic pivot introduction

Definition 6.24 (Head-corner prediction)

For every inactive edge $\langle i, j, c/S \rightarrow \alpha \rangle$ *in the chart* $\mathcal{C}$ *and all rules of the form* $\beta \rightarrow c_1/S_1 \ldots \#c_k/S_k \ldots c_n/S_n$ *where* $\#$ *marks the semantic head and* S *and* S_k *are unifiable there is an edge* $\langle i, j, \beta \rightarrow ?c_1/S_1 \ldots c/S \ldots ?c_n/S_n \rangle$ *in* $\mathcal{C}$.

See Figure 6.7.

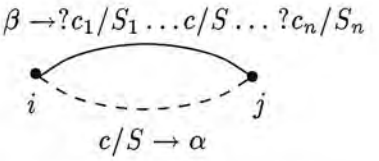


Figure 6.7: Head-corner prediction

[12] If the edges are already in the chart there is no need for them to be included again.

Definition 6.25 (Fundamental rule)

For every active edge $\langle i, j, c \rightarrow c_1/S_1 \ldots ?c_k/S_k \ldots c_n/S_n \rangle$ *and an inactive edge* $\langle w, k, \alpha \rightarrow \ldots \rangle$ *if there is a forward link* $\langle j, w \rangle$ *and* c_k/S *is unifiable with* α *then there is an edge* $\langle i, k, c \rightarrow c_1/S_1 \ldots \alpha \ldots c_n/S_n \rangle$ *in* $\mathcal{C}$.

See Figure 6.8.

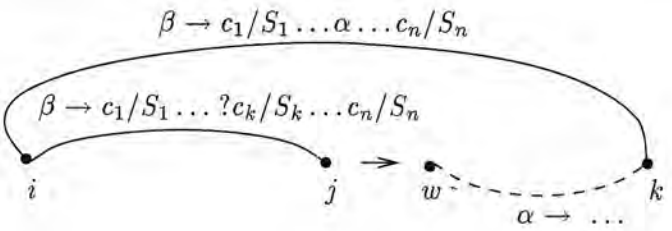


Figure 6.8: Fundamental rule

We show what the behaviour of the algorithm is using the simple grammar in Figure 6.9.

(1) s/LF → np/X, #vp(X,Y)/LF.
(2) vp(X,Y)/LF → #v(X,Y)/LF, np/Y.
(3) np/j → *john*.
(4) v(X,Y)/likes(X,Y) → *likes*.
(5) v(Y,X)/likes(X,Y) → *pleases*.
(6) np/m → *mary*.

Figure 6.9: Example grammar (SynCat/Sem → ... #head ...)

Figure 6.10 shows the state of the chart using the grammar in Figure 6.9. The input generation goal is s/likes(j,m). Only edges used for the derivation of *John likes Mary* are shown.[14]

The first inactive edge is introduced from grammar rule (4). Head-corner prediction leads to the creation of active edge 2 using grammar rule (2). Dynamic pivot introduction acting on active edge 2 introduces inactive edge 3 using grammar rule (6) and puts a forward link (L1) from the end of edge 2 to the beginning of edge 3. The fundamental rule puts together edges 2 and 3 connected with link L1 and creates inactive edge 4. Inactive edge 4 (due to head-corner prediction) leads to active edge 5. Active edge 5 (due to dynamic pivot introduction) leads to the creation of inactive edge 6 and link L3 connecting the end of edge 5 and the beginning of edge 6. The fundamental rule acting on active edge 5 and inactive edge 6 gives inactive edge 7 which is the final result.

[14] Another realisation of the same semantics is also possible—*Mary pleases John*.

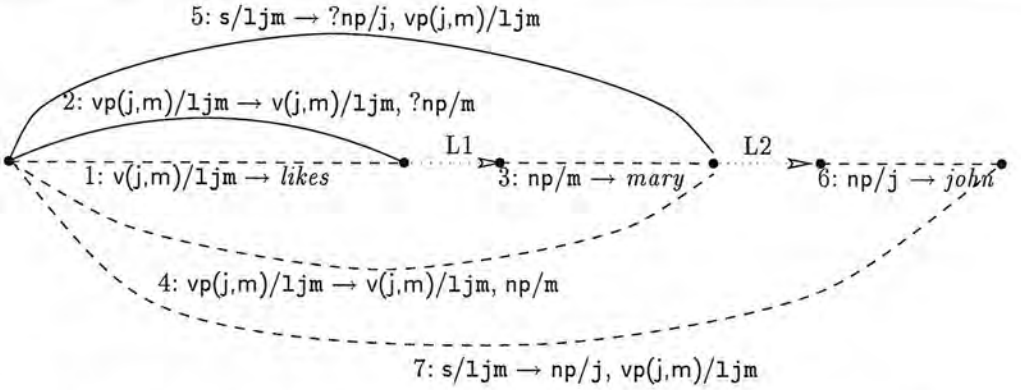


Figure 6.10: State of the chart. `ljm` abbreviates `likes(john,mary)`

The conditions for exploring the chart in a top-down manner are:

> **Procedure 1′:** For every active edge $\langle i, j, c \rightarrow c_1/S_1 \ldots ?c_k/S_k \ldots c_n/S_n \rangle$ in $\mathcal{C}$ (where c_k is the first non-expanded constituent) and all rules of the form: $a/S \rightarrow a_1 \ldots a_n$ such that S_k and S are unifiable, there is an empty active edge $\langle x, x, a \rightarrow a_1 \ldots a_n \rangle$ and a forward link $\langle j, x \rangle$.

See Figure 6.11.

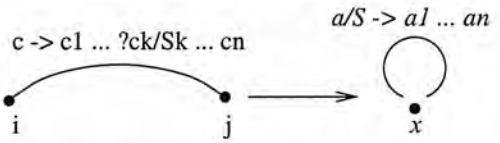


Figure 6.11: Constraints for top-down regime

However, the interpretation of edges is not very intuitive in this framework. While in parsing active edges express possible completions of a string of words a similar analogy for the presented approach cannot be made. We propose an alternative chart generation specification in Section 6.2.3.

Adjacent edges cannot be placed in a linear sequence.

6.2.3 The new formulation

In the following we present chart generation from non-hierarchical input. We believe the new formulation is simpler, more intuitive and allows for direct connections to work on parsing (which is why we introduced chart parsing earlier in Section 6.2.1). We formulate chart generation for CFGs and give an illustrative example. This section serves as a gradual introduction to the more important one on chart generation with non-concatenative grammars (Section 6.2.4).

We consider augmented context-free grammar rules where each node specifies what its semantics is. Each graph appearing in the rules has a single node ("the semantic head concept") which acts as a root. The CFG rule:

$$cat \rightarrow c_1, \ldots, c_n$$

now becomes:

$$cat/Graph - Head \rightarrow c_1/G_1 - H_1, \ldots, c_n/G_n - H_n.$$

All graphs G_i on the righthand side should be subgraphs of $Graph$. A semantic head daughter is a righthand side constituent whose head concept for its semantics is identical to the head concept of the left hand side constituent (this notion of a head daughter here is different from the one employed in [Shieber *et al.* 90] where the head daughter must have the same semantics as the mother).

If the generator uses the notion of approximate generation what constitutes success becomes different from the requirement that the generator must consume all of the semantics. We will address this later.

As later we consider non-concatenative grammars (DTGs) we will not use the notion of one level productions as they are used for context-free grammars. We will talk about the top level syntactic category and the leaf non-terminal constituents of an elementary structure (d-tree).

Definition 6.26 (Edges)

Edges are tuples $\langle Sem, Head, Struc, Goals \rangle$ where:

1. Sem is how much of the initial *Sem Graph* is covered by the edge;
2. $Head$ is the head concept of Sem;
3. $Struc$ is the syntactic structure annotated with semantic information;
4. $Goals$ is a list of remaining (generation) goals which are triples in the format $SynCat/G - H$.
5. There must be a mapping rule $Sem - Head \Leftrightarrow Synt$ in the grammar which is used to create the edge.

Inactive edges are edges with an empty list of remaining goals: $\langle Sem, Head, Syn, \{\} \rangle$

The linear notion 'to refer to a portion of the input $\langle i, j \rangle$' is now replaced with a subgraph S_i of the *Input Semantics*. The dot $\bullet$ which was used to separate the covered/expanded constituents from those expected to be covered is replaced with the notion of the remaining (generation) goals.

Initialisation: For every subgraph S of the *InputSem* which matches closely the semantics Sem of a mapping rule R with no internal generation goals $Cat/Sem - Head \rightarrow [word]$ the edge $\langle S, Head, R, \{\} \rangle$ is in the chart $\mathcal{C}$.

Fundamental Rule: If the chart $\mathcal{C}$ contains an active edge

$$\langle Sem_1, Head_1, Rule_1, \{Syn_1/Sem_1 - Head_1\} \cup Goals \rangle,$$

and an inactive edge

$$\langle Sem_2, Head_2, Rule_2, \{\} \rangle,$$

s.t. Sem'_1 matches Sem_2 and Syn'_1 is unifiable with the mother node of $Rule_2$, then there is in the chart $\mathcal{C}$ an edge $\langle Sem_1 \cup Sem_2, Head_1, Rule'_1, Goals \rangle$ where $Rule'_1$ might be a more instantiated version of $Rule_1$.

Abstractly the fundamental rule is saying the very same thing as the fundamental rule for CFG parsing: *If there is an edge that needs something and there is an edge that provides it, then the system can make progress on the first structure.*

6.2.3.1 Principles of rule introduction

Top-down: For every goal $Cat/Sem-Head$ in the remaining generation goals of any edge there is an edge in $\mathcal{C}$ $\langle Sem_e, Head, R, Goals\rangle$ if there is a rule R': $Cat'/Sem' - Head \rightarrow Goals$ and the mother syntactic category of the syntactic structure of the rule Cat' unified with Cat results in the mapping rule R' being possibly more instantiated R and the semantics of the generation goal Sem matches the semantics of the rule Sem' and Sem' covers Sem_e from the *InputSem* (by itself, not jointly with Sem).

Top-down initialisation: For every mapping rule R: $s/Sem - H \rightarrow RHS$ s.t. Sem matches the *Input Semantics* there is in the chart $\mathcal{C}$ an edge $\langle Sem', H, Synt, RHS\rangle$.

The above is equivalent to having the initial $\langle 0, 0, s \rightarrow \bullet\ c_1 \ldots c_n\rangle$ edges in top-down chart parsing.

Bottom-up: An edge $\langle Sem, Head, Rule, \{\}\rangle$ is in the chart $\mathcal{C}$ if there is a rule *Rule* $cat/S - Head \rightarrow RHS$ and there are inactive edges corresponding to all of its non-expanded constituents in RHS $\langle Sem_i, H_i, Synt_i, \{\}\rangle$ where

$$Sem = (S \bigcup_{i=1}^{n} Sem_i) \cap InputSem$$

The equivalent to the left-corner bottom-up strategy is (now a head corner one):

Head-corner: For every inactive edge

$$\langle Sem, Head, Struc, \{\}\rangle$$

there is also an edge $\langle Sem'_i, Head, R_i, RHS\_Goals_i \backslash (Cat/Sem-Head)\rangle$ for all rules R_i: $Cat_i/Sem_i - Head \rightarrow \ldots (Cat/Sem - Head)\ldots$ which share the same semantic head *Head* and whose internal generation goals (RHS) are $Goals_i$. $Sem'_i = Sem_i \cap InputSem$.

These definitions for CFG chart generation might seem somewhat abstract and we give an illustrative example in the next section.

6.2.3.2 Example for CFGs

In this section we give an example of how a sentence (*Honest men sleep soundly*) is generated using a chart which is explored top-down. The grammar is shown in Figure 6.12.

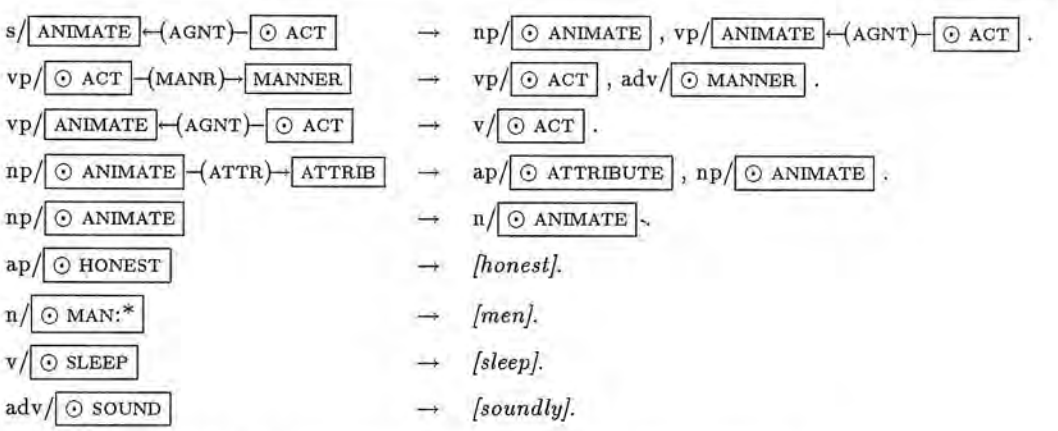


Figure 6.12: Example grammar with non-hierarchical semantics.

Concepts written like ⊙ CONCEPT: REF are the head concepts of graphs (i.e., ⊙ marks the head). A state of the chart is shown in Figure 6.13. In order to be consistent with the adopted conventions in chart parsing we draw active edges above the subgraph that they cover. Inactive edges are similarly shown below the subgraph that they cover. Generation goals are visualised as non-terminal syntactic nodes linked to the head semantic concept of the subgraph they cover (the subgraph covered by the generation goal is not shown). Edges are numbered in the order in which they were entered into the chart.

Table 6.1 gives the order in which edges are introduced into the chart and the reason why they were put in it. As can be seen from the table the interactions of an edge are considered immediately.

This example is carefully chosen to illustrate the aspect of chart generation. The more examples are considered, it becomes quickly apparent, that it is very hard to use visualisation approaches which have active edges above and inactive below the parts of the semantics they represent. This is because we no longer have the one dimensional index positions as in parsing but the two dimensional headed subgraph notion which

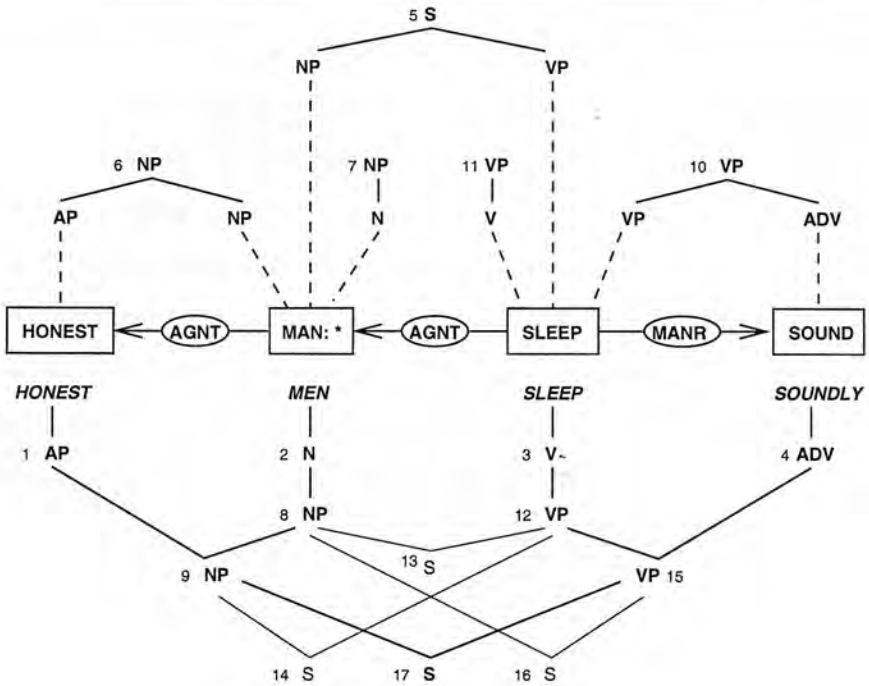


Figure 6.13: Top-down chart generation.

EDGE	RULE	TYPE	PRINCIPLE
1	$AP \rightarrow [honest]$.	inactive	Initialisation
2	$N \rightarrow [men]$.	"	"
3	$V \rightarrow [sleep]$.	"	"
4	$ADV \rightarrow [soundly]$.	"	"
5	$S \rightarrow NP,\ VP$.	active	Top-down initialisation
6	$NP \rightarrow AP,\ NP$.	"	Top-down introduction on NP in 5
7	$NP \rightarrow N$.	"	Top-down introduction on NP in 5
8	$NP \rightarrow N$.	inactive	Fundamental rule: 7 + 2
9	$NP \rightarrow AP,\ NP$.	"	Fundamental rule: 6 + (1 + 8)
10	$VP \rightarrow VP,\ ADV$.	active	Top-down introduction on VP in 5
11	$VP \rightarrow V$.	"	Top-down introduction on VP in 5
12	$VP \rightarrow V$.	inactive	Fundamental rule: 11 + 3
13	$S \rightarrow NP,\ VP$.	"	Fundamental rule: 5 + (8 + 12)
14	$S \rightarrow NP,\ VP$.	"	Fundamental rule: 5 + (9 + 12)
15	$VP \rightarrow VP,\ ADV$.	"	Fundamental rule: 10 + (12 + 4)
16	$S \rightarrow NP,\ VP$.	"	Fundamental rule: 5 + (8 + 15)
17	$S \rightarrow NP,\ VP$.	"	Fundamental rule: 5 + (9 + 15)

Table 6.1: Example order in which rules are introduced into the chart

is used to index edges.

One method of presenting the semantically indexed chart is to have (as in the case of parsing) a two dimensional array where along the row we have the head concepts and along the columns we have the sets of active and inactive edges indexed by the particular head concept on the left. Table 6.2 gives a flavour of how the semantic chart looks. The structure of the individual items is: Cat_{index}/[Cat(idx), ...], i.e., a category with a subcategorisation list. Elements on the SUBCAT list are gradually consumed (a la HPSG-style) until none are required. The *index* in Cat_{index} is needed in case the constituent's head index is not the one for the row in which the item is placed (cf. items 5,6 and 10). Inactive edges have the format: Cat/[].

VERTEX	ACTIVE	INACTIVE
h	6 np_m/[ap(h),np(m)]	1 ap/[]
m	6′ np/[np(m)] 7 np/[n(m)] 5 s_s/[np(m),vp(s)]	2 n/[] 8 np/[] 9 np/[]
s	5′ s/[vp(s)] 10′ vp/[vp(s)] 11 vp/[v(s)]	3 v/[] 12 vp/[] 15 vp/[] 13,14,16,17 s/[]
so	10 vp_s/[vp(s),adv(so)]	4 adv/[]

Table 6.2: The semantic chart

Active edge 5 combines with inactive edge 9 and according to the fundamental rule gives rise to 5′.[15] We, thus, combine items in the same row and place the items in a row for which the next element on the subcategorisation list will be indexed on. If the SUBCAT list is exhausted then we use the index of the category in order to decide where to place the new item. We will not dwell more on the visualisation of the semantic chart here.[16]

[15] In the previous chart we combined 5 simultaneously with 9 and 15 so as not to clutter the picture. Here we need to do things in 'slow motion' in order to illustrate the fundamental rule.

[16] This formulation of the semantic chart was briefly presented by M. Kay at ESSLLI'97 (one slide). We have been a bit liberal with the SUBCAT lists, i.e., we did not use s/[np(agnt),np(ptnt)], in order to have more parallels between the chart in Table 6.2 and Figure 6.13.

6.2.4 Chart generation for DTGs

In chart parsing we had dotted rules which recorded the progress that that parser had made with a CFG rule from left to right. In fact what we were interested in was what constituents remained to be parsed (for recognition). This is straightforward to modify for the case of DTGs. Instead of CFG productions we have d-trees and instead of remaining constituents to the right we can have a set of remaining constituents to be generated. Because we are interested in generation we are no longer tied to the adjacency constraints for parsing.

Here edges will contain similar information as in the case for CFGs:

Definition 6.27 (DTG Edges)

Edges are tuples $\langle Sem, Head, MR, Goals \rangle$ where:

1. Sem is how much of the initial semantics (*InputSem*) is covered by the edge;
2. $Head$ is the head concept of Sem;
3. MR is a mapping rule (i.e., a syntactic structure (d-tree) annotated with semantic information);
4. $Goals$ is a list of remaining (generation) goals which are triples in the format $SynCat/G - H$.

The abstract notion of an edge and its use is almost identical to the CFG case. The differences lie in the fact that the syntactic-semantic structures are now d-trees and in order to combine two structures we again will be interested in the root nodes of the structures and also in the other projection nodes which might happen to be internal. The projection nodes are the nodes that are identified with the complementation nodes during subsertion (see Section 4.3.2).

6.2.4.1 Inference rules

Initialisation:

For every subgraph S of the input semantics *InputSem* which matches closely a lexical mapping rule MR (i.e., a rule with no internal generation goals) $\langle Sem, Head, MR, \{\} \rangle$ is in the chart $\mathcal{C}$. $Head$ must be in *InputSem* and Sem is the maximal projection of the applicability semantics of MR and *InputSem*.

Strictly speaking the initialisation stage is not necessary for top-down processing. Yet, in a truly lexicalised grammar this stage will give all possible mapping rules that will need to be considered (of course allowing for non-lexical but lexicalised! rules). What we have in mind above is the equivalent of setting up the preterminal categories in parsing.

We consider two stages of processing (performed in sequence): (i) skeleton building, and (ii) covering the remaining semantics (cf. Section 5.3) and the fundamental rules for these stages differ slightly:

Fundamental Rule (complementation stage):
If the chart $\mathcal{C}$ contains an active edge $\langle S_1, Head, MR, \{Cat'_1/S'_1 - H_1\} \cup Goals\rangle$, and an inactive edge $\langle S_2, H_2, MR_2, \{\}\rangle$, s.t. S'_1 matches S_2 (joining H_1 and H_2) and Cat'_1 is unifiable with $syn(proj\_node(MR_2))$,[17] then there is in $\mathcal{C}$ an edge $\langle S_1 \cup S_2, Head, MR', Goals\rangle$ where MR' might be more instantiated than MR due to the unification of the syntactic nodes and the maximal join between the semantic structures.

Fundamental Rule (modification stage):
If the chart $\mathcal{C}$ contains a modifying inactive edge $\langle S, H, MR, \{\}\rangle$ and a modified edge (active or inactive) $\langle S_1, H_1, MR_1, Goals\rangle$ s.t. MR_1 contains a node N with semantic annotation $S_N - H_N$ (graph-head) and sister adjunction constraint $\langle LR, Syn_N\rangle$ and $S - H$ matches $S_N - H_N$, MR is consistent with Syn_N, then there is in $\mathcal{C}$ an edge $\langle S_1 \cup S, H_1, MR'_1, Goals\rangle$. Again MR' is possibly the more instantiated version of MR_1 due to the matchings on the syntactic and semantic sides.

Top-down initialisation:
For every mapping rule MR whose applicability semantics is $Sem - H$ (graph-head) s.t. Sem matches the input semantics $InputSem$ there is in the chart $\mathcal{C}$ an edge $\langle Sem', H, MR, Goals\rangle$ where $Goals$ is the set of internal generation goals of MR (the leaf non-terminal nodes of MR with their linkage to the semantics).

Top-down prediction (complementation stage):

[17] $syn(proj\_node(MR_2))$ refers to the syntactic part of one of the maximal projection nodes of the d-tree of mapping rule MR_2.

For every goal $Cat/Sem - Head$ in the remaining generation goals of any edge there is an edge in the chart C $\langle Sem_e, Head', MR', Goals \rangle$ if there is a mapping rule MR with applicability semantics $Sem_2 - Head_2$ (graph-head) and $Head$ and $Head_2$ can be joined resulting in $Head'$, Cat unifies with $syn(proj\_node(MR))$, Sem matches Sem_2 and Sem_2 covers Sem_e from the input semantics $InputSem$ (by itself, not jointly with Sem). Again MR' is possibly the more instantiated version of MR_1 due to the matchings on the syntactic and semantic sides.

Bottom-up prediction:
An edge $\langle Sem, Head, MR', \{\} \rangle$ is in the chart C if there is a mapping rule MR with applicability semantics $S - Head$ (graph-head) and there are inactive edges corresponding to all of its non-expanded constituents $\langle Sem_i, H_i, MR_i, \{\} \rangle$ where

$$Sem = (S \bigcup_{i=1}^{n} Sem_i) \cap Input\ Semantics.$$

The bottom-up prediction is very similar indeed to head corner bottom-up parsing [vanNoord 93].

The details of our implementation of the chart generation are based on an implementation of a CFG chart parser by Chris Mellish [Mellish 95].

Prediction The edge that is being added is active (it has a non-empty set of generation goals), but there is no edge for it to combine with already in the chart. In this case the required edges are predicted, by entering an initial active edge for each way they can be built.

Scanning The edge that is being entered is active, and there are edges with which it can combine. For each of these edges the fundamental rule is applied, giving rise to further edges.

Completion The edge that is being entered is complete. For each of the active edges that it can combine with, the fundamental rule is applied, giving rise to further edges.

We consider agenda-based processing in the next chapter.

6.2.4.2 Recovering sentence structure

We use a technique for encoding a generation forest (which is similar to a parse forest). Edges are annotated with information about which other edges lead to their creation and how. Derivations are enumerated top-down by first locating an edge that consumes all the input semantics (or in the approximate generation case a sufficient part of it). Then the links to the 'creators' (similar to the list of daughters in the parsing forests for CFG grammars) of the edge are recursively followed and their d-trees are recovered. Once these are computed the two d-trees involved in the creation of the first edge that was considered are put together giving rise to the final d-tree. This d-tree (description of a tree) is converted to a tree by closing off all d-links.

In recursively exploring the creators of edges, not all descriptions in all edges are taken into account. Intermediate and inactive edges do not contribute structural material. Only lexical edges and edges that have been introduced by top-down prediction or modification do. The actual type of the edge is expressed in the 'creators' field. In cases where the edge was put in the chart because of two other edges we also keep the nodes in the descriptions of the trees corresponding to these edges that will be identified when the two descriptions are put together. This information was computed anyway while deciding wether the two edges can combine. The information is indeed recoverable but having these node explicitly represented allows for very efficient enumeration of the derivations. We will see examples of the annotations that go in the 'creators' field and actual recovering of generated trees in the next section.

6.2.5 Example

In this section we show an example of the trace of the generation of the sentence *Alexander launched a full-scale attack on Athens.*

For the purposes of the example a feature structure:
$\textsf{Cat}\begin{bmatrix}\textsc{attr}_1\textsc{:}\ \textsc{val}_1\\ \textsc{attr}_2\textsc{:}\ \textsc{val}_2\end{bmatrix}$ will be written as `CAT(ATTR1:VAL1, ATTR2:VAL2)`

In the follwoing we will be visualising edges by showing their:

- identifier (`Id` field);
- the parts of the input semantics they consume (`Cnsmd` field);
- the list of maximal projections (`MaxPs` field). each element is a triple of syntactic node, feature structure and semantic handle;
- d-tree (`Syn` field). We will display the trees graphically. We will need to identify some nodes by their name (and not just their labels.
- list of internal generation goals (`Goals` field). This is a list of elements in the same format as in the `MaxPs` field. An empty `Goals` list means that the edge is an inactive one.
- information about why the edge is in the chart (`Created` field).

```
twain[~nicolas/protector] protector

>>> PROTECTOR loaded <<<

| ?- chart_generate 130.                % generate example number 130

SEMANTICS =
  [ attack(a,_),
    commander(c,'Alexander'),
    town(t,'Athens'),                   %  [PERSON: Alexander]<-AGNT-[ATTACK]-OBJ->[TOWN: Athens]
    agnt(a,c),                          %                                  |
    obj(a,t),                           %                                MANR
    full_scale(f,_),                    %                                 V
    manr(a,f)                 ]         %                           [FULL_SCALE]
```

The goal is to realise the above semantics as a sentence (S). The first edge that is added in the chart is a transitive construction. It is chosen in the top-down initialisation (the value of the `Created` field is `[top-down]`) because it has an applicability semantics which matches the input semantics above, and the maximal projection and root of its syntactic part unifies with our syntactic goal.

```
### Entering edge top-down
------------------------------------------------------------------------
Id   : 1
Cnsmd: a:[agnt(a,c),obj(a,t)]
MaxPs: [s(1):s(fin:yes):a]
Syn  :
                    s
              +---------+
              |         vp
              |      +-----+
npO(1) ->   np       v     np     <- np1(1)
                     ^
                     |
                    v(1)

Goals  : [v(1)  : v(vform:fin): a,
          npO(1): np(case:nom): c,
          np1(1): np(case:acc): t]
Created: [top-down]
```

This is an active edge. It needs to fill in the preterminal position (node `v(1)`), and to generate its subject (node `npO(1)`) and object (node `np1(1)`). The nodes of the tree (not their lables) (`np1(1)`, `v(1)`, `npO(1)`, etc.) have an index which in this case is the same as the identifier of the edge (`Id`). These are not always the same. They can differ when a new edge (edge identifier is increased) is usig the d-tree of a previous edge but just reducing the number of elements on the `Goals` list.

Edge 1 predicts that it needs a tree that can be subserted at node `v(1)`, i.e., a tree with root/maximal projection `v(vform:fin)` and semantics with a head handle `a` (the concept ATTACK). A structure that satisfies these requrements is a lexical definition which is entered in the chart.

```
### Prediction for edge . . .  : 1
### Entering edge predict:1/v(1)
------------------------------------------------------------------------
Id   : 2
Cnsmd: a:[attack(a,_)]
MaxPs: [nd(2):v(vform:fin,lex:attack):a]
Syn  :
        v  <- nd(2)
        |
      attack

Goals  : []
Created: [predict:1/v(1)]
```

In contrast to the top-down generation we do not combine the two structures together. Edge 2 simply says that it realises the concept ⊙ ATTACK. It also says in the `Created` field that its syntax can be incorporated with the syntactic structure in edge 1 at node `v(1)`. The actual incorporation happens in the phase of enumerating the derivation

structures.

Edge 2 is an inactive edge and can combine with any other edge looking for it. Having been predicted by edge 1 it is natural that they can combine resulting in edge 3 which is like edge 1 but with one less generation goal. Also note how the consumed semantics includes the semantics of both 'creator' edges.

```
### Completions for edge . . . : 2
### Entering edge complete:1/v(1):2
----------------------------------------------------------------------
Id   : 3
Cnsmd: a:[attack(a,_),agnt(a,c),obj(a,t)]
MaxPs: [s(1):s(fin:yes):a]
Syn  :
                    s
             +------------+
             |            vp
             |        +--------+
             |        v        |
             |        |        |
np0(1) ->    np     attack     np   <- np1(1)

Goals  : [np0(1):np(case:nom):c,
          np1(1):np(case:acc):t]
Created: [complete:1/v(1):2]
```

Edge 3 is also an edge where we have examples of the identifier of the edge and the index of the nodes in the d-tree being different. Edge 3 represents an elementary structure as defined in the mathematical framework of D-Tree grammars — the lexical anchor is in place. Edge 3 (just like edge 1) still needs these two NPs for the subject and object of the construction. So it predicts the first.

Edge 4 is interesting because its consumed semantics is empty. In a way it is there just to add some syntactic structure.

```
### Prediction for edge . . .  : 3
### Entering edge predict:3/np0(1)
----------------------------------------------------------------------
Id   : 4
Cnsmd: c:[]
MaxPs: [np(3):np(case:nom):c]
Syn  :
         np
         |
         pn   <- pn(3)

Goals  : [pn(3):pn(...):c]
Created: [predict:3/np0(1)]
```

Edge 5 is introduced because our current goal matches a lexical definition.

```
### Prediction for edge . . .  : 4
### Entering edge predict:4/pn(3)
----------------------------------------------------------------------
Id    : 5
Cnsmd: c:[commander(c,'Alexander')]
MaxPs: [nd(4):pn(num:sg,lex:'Alexander'):c]
Syn   :
          pn  <- nd(4)
          |
      Alexander

Goals  : []
Created: [predict:4/pn(3)]
```

Edge 6 represents returning to the vacuous edge 4 and making it inactive.

```
### Completions for edge . . . : 5
### Entering edge complete:4/pn(3):5
----------------------------------------------------------------------
Id    : 6
Cnsmd: c:[commander(c,'Alexander')]
MaxPs: [np(3):np(case:nom)):c]
Syn   :
          np  <- np(3)
          |
          pn

Goals  : []
Created: [complete:4/pn(3):5]
```

Note that while the linguistic knowledge base (grammar) does contain other mapping rules for NP they do not apply because they are not semantically licensed. The referent of the concept PERSON: ALEXANDER, i.e., the name Alexander can only license the proper name construction. Here we see the the conditions associated with mapping rule put to use. Now that we know how to generate at least one subject for edge 3 we can concentrate on generating the object NP: edge 7 is like edge 3 but has one generation goal less and it also has consumed more of the input semantics.

```
### Completions for edge . . . : 6
### Entering edge complete:3/np0(1):6
----------------------------------------------------------------------
Id    : 7
Cnsmd: a:[commander(c,'Alexander'),attack(a,_),agnt(a,c),obj(a,t)]
MaxPs: [s(1):s(fin:yes):a]
Syn   :
                      s              <- s(1)
              +-------------+
              |             vp
              |          +--------+
              |          v        |
              |          |        |
np0(1) ->     np     attacks     np   <- np1(1)

Goals  : [np1(1):np(case:nom):t]
Created: [complete:3/np0(1):6]
```

Edges 8–10 represent generating the object NP are identical to edges 4–6.

```
### Prediction for edge . . .  : 7
### Entering edge predict:7/np1(1)
----------------------------------------------------------------------
Id   : 8
Cnsmd: t:[]
MaxPs: [np(5):np(case:acc):t]
Syn  :
       np    <- np(5)
       |
       pn

Goals  : [pn(5):pn(...):t]
Created: [predict:7/np1(1)]
### Prediction for edge . . .  : 8
### Entering edge predict:8/pn(5)
----------------------------------------------------------------------
Id   : 9
Cnsmd: t:[town(t,'Athens')]
MaxPs: [nd(6):pn(num:sg,lex:'Athens'):t]
Syn  :
       pn   <- nd(6)
       |
     Athens

Goals  : []
Created: [predict:8/pn(5)]
### Completions for edge . . . : 9
### Entering edge complete:8/pn(5):9
----------------------------------------------------------------------
Id   : 10
Cnsmd: t:[town(t,Athens)]
MaxPs: [np(5):np(num:sg,per:3,case:nom):t]
Syn  :
       np
       |
       pn

Goals  : []
Created: [complete:8/pn(5):9]
```

Thus far, we know how to generate all complements of the original construction that we found. So this construction can be made into an inactive edge.

```
### Completions for edge . . . : 10
### Entering edge complete:7/np1(1):10
----------------------------------------------------------------------
Id   : 11
Cnsmd: a:[town(t,'Athens'),commander(c,'Alexander'),attack(a,_),
          agnt(a,c),obj(a,t)]
MaxPs: [s(1):s(fin:yes):a]
Syn  :
              s          <- s(1)
     +------------+
     |            vp
     |         +-------+
     |         v       |
     |         |       |
     np    attacks    np

Goals  : []
Created: [complete:7/np1(1):10]
```

If we were working in scenarios with time-pressure (time-outs) the system could allow interruption at this stage and if need be it can quickly recover a skeletal structure like: *Alexander attacks Athens.*

As an alternative ot the very first mapping rule we chose we could consider an alernative which involves a light verb (*launch an attack*). Edge 12 is predicted from the initialisation.

```
### Entering edge top-down
------------------------------------------------------------------------
Id    : 12
Cnsmd: a:[attack(a,_),agnt(a,c),obj(a,t)]
MaxPs: [s(7):s(fin:yes):a]
Syn   :
                              s          <- s(7)
             +----------------------+
             |                      vp
             |       +--------------------------+
             |       v              np              pp
             |       |        +-------+       +-----+
             |       |      det      nbar     p     |
             |       |       |        n       |     |
             |       |       |        |       |     |
npO(7) ->    np     launch   an      attack   on    np   <- np1(7)

Goals  : [npO(7):np(case:nom):c,
          np1(7):np(case:acc):t]
Created: [top-down]
```

While structurally different from the d-tree in edge 1, edge 12 is still going to look for the same complements: a nominative NP with a handle c (commander) and a accusative NP with handle t (town). These have already been found previously and all we need to do is update the Created fields to reflect that the NP edges are also needed by the new alternative construction. Here is the update for the edge corresponding to the subject NP:

```
### Prediction for edge . . .  : 12
### Updating edge 4:
------------------------------------------------------------------------
Id    : 4
Cnsmd: c:[]
MaxPs: [np(3):np(case:nom):c]
Syn   :
         np
         |
         pn   <- pn(3)

Goals  : [pn(3):pn(...):c]
Created: [predict:3/npO(1),predict:12/npO(7)]
```

Edge 13 reflects the fact that we know how to deal with the subject.

```
### Entering edge complete:12/np0(7):4
------------------------------------------------------------------------
Id   : 13
Cnsmd: a:[commander(c,'Alexander'),attack(a,_),agnt(a,c),obj(a,t)]
MaxPs: [s(7):s(fin:yes):a]
Syn  :
                              s          <- s(7)
              +-----------------------+
              |                      vp
              |        +--------------------------+
              |        v              np            pp
              |        |        +-------+      +-----+
              |        |       det     nbar     p     |
              |        |        |       n       |     |
              |        |        |       |       |     |
npO(7) ->     np    launches   an    attack    on    np   <-np1(7)

Goals  : [np1(7):np(case:acc):t]
Created: [complete:12/np0(7):4]
```

Edge 13 will predict we will be looking for the object NP which we already know how to find. So we update the corresponding edge to include the information that its structure will be used by the current edge (13).

```
### Prediction for edge . . .  : 13
### Updating edge 8: predict:13/np1(7)
------------------------------------------------------------------------
Id   : 8
Cnsmd: t:[]
MaxPs: [np(5):np(case:acc):t]
Syn  :
       np    <- np(5)
       |
       pn

Goals  : [pn(5):pn(...):t]
Created: [predict:7/np1(1),predict:13/np1(7)]
```

Now we have all that was needed for the alternative top-level construction: we add an inactive edge.

Edge 13 reflects the fact that we know how to deal with the subject.

```
### Entering edge complete:12/np0(7):4
------------------------------------------------------------------------
Id   : 13
Cnsmd: a:[commander(c,'Alexander'),attack(a,_),agnt(a,c),obj(a,t)]
MaxPs: [s(7):s(fin:yes):a]
Syn  :
                              s          <- s(7)
              +-------------------------+
              |                        vp
              |         +---------------------------+
              |         v              np             pp
              |         |         +-------+      +-----+
              |         |        det     nbar     p     |
              |         |         |       n       |     |
              |         |         |       |       |     |
npO(7) ->     np    launches    an    attack    on    np   <-np1(7)

Goals  : [np1(7):np(case:acc):t]
Created: [complete:12/np0(7):4]
```

Edge 13 will predict we will be looking for the object NP which we already know how to find. So we update the corresponding edge to include the information that its structure will be used by the current edge (13).

```
### Prediction for edge . . .  : 13
### Updating edge 8: predict:13/np1(7)
------------------------------------------------------------------------
Id   : 8
Cnsmd: t:[]
MaxPs: [np(5):np(case:acc):t]
Syn  :
       np     <- np(5)
       |
       pn

Goals  : [pn(5):pn(...):t]
Created: [predict:7/np1(1),predict:13/np1(7)]
```

Now we have all that was needed for the alternative top-level construction: we add an inactive edge.

```
### Entering edge complete:13/np1(7):10
------------------------------------------------------------------------
Id    : 14
Cnsmd: a:[town(t,'Athens'),commander(c,'Alexander'),attack(a,_),
          agnt(a,c),obj(a,t)]
MaxPs: [s(7):s(fin:yes):a]
Syn   :
                          s
      +-------------------------+
      |                        vp
      |         +---------------------------+
      |         v             np            pp
      |         |          +-------+      +-----+
      |         |         det     nbar     p     |
      |         |          |       n       |     |
      |         |          |       |       |     |
      np     launches     an     attack    on    np

Goals  : []
Created: [complete:13/np1(7):10]
```

This concludes the top-down phase when initial mapping rules are chosen and their complements are explored (but not combined) recursively. In phase 2 the system augments, extends existing edges by combining them with modification constructions which are semantically licensed.

```
### PHASE 2: incorporating modifiers ################################

### Entering edge modification
------------------------------------------------------------------------
Id    : 15
Cnsmd: a:[full_scale(full_scale,_),manr(a,full_scale)]
MaxPs: [nbar(8):nbar:a]
Syn   :
        nbar  <- nbar(8)
         |
         a
         |
     full-scale

Goals  : []
Created: [modification]
```

For explanatory purposes the mapping rule on which edge 15 is based, has been made more specific that it really is. In particular the lexical anchor *full-scale* has already been specified in the mapping rule rather than having it being generated through an internal generation goal. Figure 6.14 what the modification mapping rule.

Note that the mapping rule in Figure 6.14 does not consume the $\boxed{\odot \text{ ATTACK}}$ concept (it is not in the `Cnsmd` field of edge 15).

The inactive modification edge 15 can combine with the inactive skeletal edge 14 producing edge 16 which covers all of the semantics.

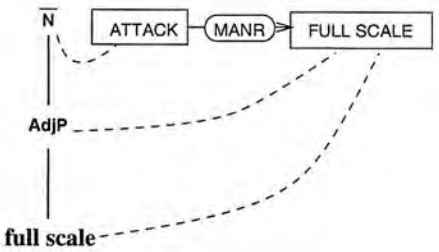


Figure 6.14: Adjectival construction

```
### Entering edge mod(pre):14/nbar(7):15/nbar(8)
------------------------------------------------------------------------
Id    : 16
Cnsmd: a:[full_scale(full_scale,_),manr(a,full_scale),town(t,'Athens'),
          commander(c,'Alexander'),attack(a,_),agnt(a,c),obj(a,t)]
MaxPs: [s(7):s(fin:yes):a]
Syn   :
                          s       <- s(7)
     +-------------------------+
     |                         vp
     |        +----------------------------+
     |        v              np             pp
     |        |          +--------+      +------+
     |        |        det      nbar     p      |
     |        |         |         n      |      |
     |        |         |         |      |      |
     np    launches    an     attack    on     np

Goals  : []
Created: [mod(pre):14/nbar(7):15/nbar(8)]
```

The `Created` field of edge 16 specifies that when the d-trees corresponding to edge 14 and edge 15 are put together (using sister adjunction) nodes `nbar(7)` and `nbar(8)` will be identified.

Unpacking the generation forest starting from edge 16 produces the following (single) solution:

```
                                    s
      +-----------------------------------+
      np                                  vp
      |          +-----------------------------------------+
      pn         v                     np                    pp
      |          |          +------------------+         +-------+
      |          |        det                 nbar       p       np
      |          |         |          +-------------+     |       |
      |          |         |          a             n     |       pn
      |          |         |          |             |     |       |
  Alexander   launches     a     full-scale      attack    on    Athens
```

```
Solutions.....: 1
Time..........: 540msec
Generation time: 28msec = 19 + 9
Mapping rules..: 8
Edges.........: 16
```

The example took 540 msec to be generated (including the display of the tracing information). If the system tracer is switched off the example is generated in 28 msec! 19 msec of these were spent generating the chart and 9 msec enumerating the derivation and converting the tree description into a tree.

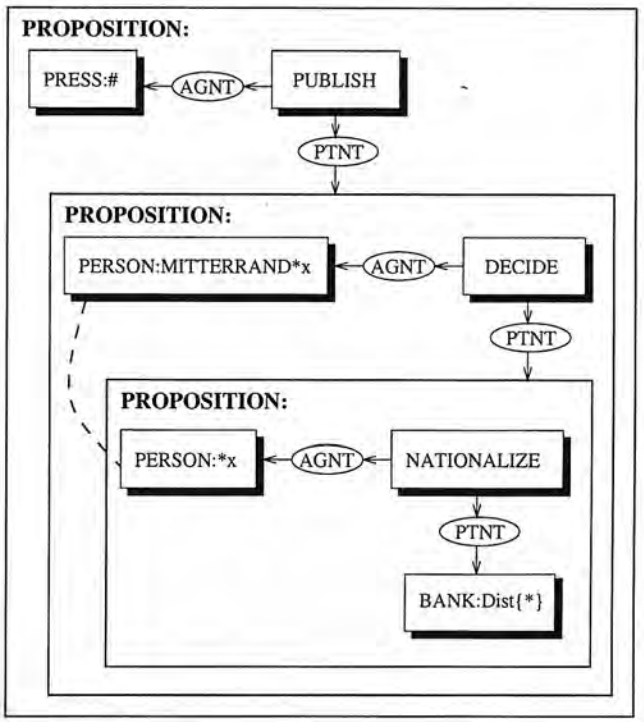


Figure 6.15: Input semantics for the Mitterrand example

6.2.6 How to express/impress

In this section we present some of the structures (including partial results) that are produced during the generation of the semantics shown in Figure 6.15. This example is from [Zock 90] where it is discussed from a psycholinguistic perpective. (...) represents syntactically incomplete sentences.

- Mitterrand nationalizes the banks.

- The nationalization of the banks by Mitterrand (...)
- The banks are nationalized by Mitterrand.
- The banks, nationalized by Mitterrand (...)
- The nationalization decided by Mitterrand (...)
- The nationalization is decided by Mitterrand.
- Mitterrand decides to nationalize the banks.
- Mitterrand decides on the nationalization of the banks.
- The decision of Mitterrand to nationalize the banks (...)
- The nationalization of the banks decided by Mitterrand (...)
- The nationalization of the banks is decided by Mitterrand.
- The banks, nationalized by Mitterrand, (...)
- The press publishes the decision of Mitterrand.
- The press publishes Mitterrand's decision.
- The press's publication of Mitterrand's decision (...)
- The publication by the press of Mitterrand's decision (...)
- The publication by the press that Mitterrand decides (...)
- The decision of Mitterrand, published by the press (...)
- The decision of Mitterrand is published by the press.
- The decision on the nationalization by Mitterrand is published by the press.
- The nationalization decided by Mitterrand is published by the press.
- The press publishes that Mitterrand decides to nationalize the banks.
- The press publishes Mitterrand's decision to nationalize the banks.
- The publication of Mitterrand's decision to nationalize the banks (...)
- The decision of Mitterrand to nationalize the banks is published by the press.
- The decision made by Mitterrand to nationalize the banks is published by the press.
- Mitterrand decides to nationalize the banks.
- Mitterrand decides on the nationalization of the banks.
- Mitterrand's decision to nationalize the banks is published by the press.
- The nationalization of the banks decided by Mitterrand is published by the press.

- The nationalization of the banks that Mitterrand decides is published by the press.
- Mitterrand's nationalization of the banks (...)
- The banks are nationalized by Mitterrand.
- The banks nationalized by Mitterrand.

6.3 Discussion

At the beginning of this chapter we observed that one of the reasons for backtracking was the existence of lexical and syntactic gaps. But why are there lexical and syntactic gaps? In a computational setting in the context of multilingual generation this is due to the fact that in the interlingua[18] any distinction which can be made by any of the languages involved must be represented and disambiguated. Most of the generation systems consider a limited subset of the language (the grammatical knowledge of even the largest generation systems is very small compared to the real language). In a well constrained domain it is easier to avoid such pitfalls. Interestingly enough, it is again due to the same reason (insufficient coverage) that generators are likely to encounter dead ends like in examples 6.2–6.5. We are not concerned in this thesis with the question of why gaps exist in the language *per se*.

When one has a model where any change in the syntax implies a change in the semantics, the difference/motivation for non-determinism in the variation produced is much reduced. Our stance is that it is very hard (if not impossible) to describe every syntactic variation without starting to introduce semantic features directly related (perfectly aligned) to these syntactic choices (in effect this is the approach taken by Systemic Grammar) and in a large number of engineering applications the input would not be as specific as required for NLG. Underspecified input is still not an argument for non-deterministic models of processing. It is possible to augment the input with appropriate defaults, which would then be expressible by a generation system [O'Donnell 96]. If the current augmented input specification cannot be expressed for one reason or another the original input could be augmented differently and this could be fed into the

[18] That is if one is using an interlingual representation. A multilingual grammar does not necessarily commit to an interlingua.

generator afresh. This is the notion of re-generation to which we referred earlier (see page 18). In this thesis we consider the augmenting of the original input on a need be basis and control it by the upper and lower semantic bounds.

6.3.1 The space of chart generators

In describing chart generation we have been making reference to some parameters whose different instantiations give rise to the following space of chart generators:

ways of exploring the chart: bottom-up, top-down, head-driven.

order of generating constituents: left to right, right to left, any, parallel. This is related to bidirectional chart parsing [Steel & DeRoeck 87, Mellish *et al.* 94].

semantic covering: complete and coherent vs. approximate matching.

6.3.2 Kay's complexity argument for CFGs

A structure (say an NP) with n modifiers would lead to $\sum_{i=0}^{2^n} i!$ items in the chart only one of which is acceptable (the NP with all the n modifiers) an observation made by M. Kay in [Kay 96]. This is a serious problem for CFG chart generation because one doesn't want the other $\sum_{i=0}^{2^n} i! - 1$ edges to start interacting with other edges in the chart. For our framework this is not a problem as we consider the syntactic structures to be *partial*. In fact in the first stage we don't even consider modifiers. We start elaborating on modifiers after the skeletal structure has been built. Due to the extensions we have added to DTGs regarding the order of modifiers we don't have to decide on the order of modifiers for the items in the chart (all that happens when we read off (recover) the generated sentence). Although the complexity for our case is lower we still have multiple edges rooted at the distinguished category which may not cover all the input.

The computation terminates because there are a finite number of initial edges that can be put in the chart and each interaction of edges strictly consumes more semantics and leads to an edge that covers a subgraph of the input semantics and there are a finite number of such subgraphs.

In [Norvig 91] Norvig considers automatic memoization for CFG parsing. One can consider doing the same for generation (in fact Norvig's method is general enough to handle any memoization). However, the table search can be extremely expensive computationally so much so that it dominates the computation. Undoubtedly it is quicker to recompute the answers to very simple goals than it is to construct and maintain the tables (charts) needed to memoize their solutions. But unfortunately very little is known about the trade-offs involved in memoization versus recomputation in the general case.[19]

Chart generation has received some attention in the past [Shieber 88, Haruno *et al.* 93, Pianesi 93, Gerdemann & Hinrichs 95, Kay 96, Nicolov *et al.* 97]. We believe there is a lot more to be gained from considering parallels between parsing and generation in the area of using memoization techniques. This thesis is a modest contribution in that direction and undoubtedly more research will be done in this area in the future.

In the language learning literature it has been noted that learners reuse (what we would call) successfully generated constituents in false starts and when they encounter a dead-end and need to replan the structure of the sentence [Ellis 94, Ellis 97, Skehan 98]. It would be interesting to see if there are any connections that can be made between second language learners' performance and memoization-based strategies. This aspect has not received much attention in the generation literature.

6.4 Conclusion

In this chapter we have explained the inefficiencies of backtracking generators. We have motivated the use of memoization-based generation and have presented chart-based generation for non-concatenative grammars.

The fact that well-formed constituents discovered on one branch of the search space are lost if that branch eventually fails and may subsequently be recomputed leads to great inefficiency. We presented an example (*Alexander launched a full-scale attack on the town*) whose generation required backtracking due to a lexical gap. All the

[19] For some initial ideas in parsing see [vanNoord 97]. Yet again how that transfers to generation is still an open issue.

work of generating constituents prior to noticing the lexical gap will be repeated in the subsequent attempt to use the *X launched an attack on Y* construction. Because the results of an unsuccessful search are not stored they must be recomputed. The failure to store well-formed structures leads to a complexity of generation that is exponential in the size of the input semantics and for large semantics the penalty in wasted time is unacceptable.

There are direct parallels to this problem in parsing and we have considered the solution adopted there—the use of a well-formed substring table or a chart to store fully parsed constituents. We briefly looked at what charts are and how they are used in parsing. We did a rational reconstruction of an early attempt to use charts in generation which was still trying to use vertices in a similar fashion as string positions are used in parsing. We presented an alternative semantic-indexed chart generation technique initially for CFGs (and gave a reasonably detailed example) and later we extended the model by considering non-concatenative grammars (DTGs). The semantic chart associated information with portions of the input semantics. We developed equivalents of dotted rules and specified what edges in the context of generation mean. We formulated the combination rules for DTG chart generation and revisited the example *Alexander launched a full-scale attack on the the town* this time showing step by step what computations take place.

SUMMARY

- Chart generation avoids duplication of work. It is similar to chart parsing but guided by the semantics.
- Chart generation can be coupled with non-concatenative grammars too in a much simpler way than the counterpart techniques in parsing.
- Chart generation is not necessarily only bottom-up. Alternative generation strategies can be considered by using different principles for introducing items in the chart.

The generation algorithm can give multiple solutions for a given input semantics. But what do we do with these? In some application domains we might want to prefer some

paraphrases over others or even better try to generate the best one(s) first. Indeed, even with a chart a search for all solutions is likely to be prohibitively slow in almost all cases. With the chart we have eliminated some redunduncy from the search space, but we still have the original space, bigger than the one for conventional generation. We look at ways of doing preference-based generation in the next chapter.

Chapter 7

Preference-Based Approximate Generation

" 'Rule Forty-two. All persons
more than a mile high to leave the court.' ...
'Well, I shan't go,' said Alice; 'Besides
that's not a regular rule: you just invented it now.'
'It's the oldest rule in the book,' said the King.
'Then it ought to be Number One,' said Alice."
—Lewis Carroll

The way we have defined our generation model implies a certain search space. For example, given an input semantics a number of paraphrases (solutions) will be valid and the generator has to 'find' them (by exploring the search space). In the previous chapter we looked at the search space and identified that there were redundancies in it. We had a solution which converted our search space from a tree into a graph. This technique saved us work, yet we are still left with the same number of solutions we had to find (i.e., we did not sacrifice completeness). Because we have relaxed some assumptions (semantic completeness and coherence) we have a much larger search space than current generation systems. The bigger search space means we have more alternatives to explore and (what was our goal) more solutions (paraphrases). However, some sentences might be preferable to others and it is desirable for the generator to be able to produce these earlier. Thus, we want to have a generation system whose behaviour we can tune. In linguistics this is known as the issue of *performance*; in automated reasoning/logic programming as *control*. And the kind of behaviour we

want is such that the generator finds 'better' sentences earlier. This is a scenario which is common in AI and often comes under rubrics like best-first search (in the case where there is one best solution) or more generally preference-based processing. In previous research the focus has been more on building generators with increased paraphrasing capabilities—the issue of choosing alternative realisations has received little attention. This chapter is about:

1. What does it mean for a (partial) realisation to be better than another? and
2. How can we incorporate notions of 'betterness' in the generation process?

The different paraphrases will differ in a number of dimensions. We assume these differences can be captured on the representational level, i.e., that the representations of the paraphrases can be different. We first discuss how sentences can be evaluated (off-line). A number of criteria might be used: semantic, syntactic, etc. When considering semantic criteria the role of the input semantics is highlighted: valid sentences for our generation model are those whose corresponding semantics is constrained by the lower and upper semantic bounds and the input semantics in the manner described in Section 5.1.3 but now we look into ways of defining a 'better' sentence in terms of one whose corresponding semantics also deviates less from the input semantics. One way this can be done is to devise a measure of a semantic distance between two conceptual graphs and compare each generated sentence's semantics with the input semantics. We review a number of previously used techniques to perform this task but none of these seem to give satisfactory results in our case. We propose a new way of comparing two conceptual graphs corresponding to generated paraphrases with respect to the input semantics.

We also suggest a number of syntactic criteria for preferring one sentence over another. The general state of the art in this field doesn't have much to offer and we necessarily have to be more speculative.

We look at ways for integrating preferences into the generation process. To this end we investigate the use of an *agenda* with the processing (memoization) model developed in the previous chapter so that a best-first search can be performed.

In the discussion we consider some alternatives to the preference approach we have presented—namely, a classification model of the mapping rules that map the semantics onto the syntax. This classification model is based on the generalisation/subsumption hierarchy over the semantic structures. We also mention an approach for compiling syntactic patterns as it crucially depends on good techniques for performing preference generation.

Overview

We first motivate preference-based generation in Section 7.1. Structures in generation can be evaluated according to different criteria and we mention some in Section 7.2. Section 7.3 is devoted to developing a comparison model between the corresponding semantics of two paraphrases with respect to the input semantics. We justify the need for syntactic comparison in Section 7.4. We develop an integrated model for using preferences and the chart technique in Section 7.5. We discuss alternatives to some of the mechanisms in Section 7.6.

7.1 Motivation

It might not be obvious at all that the issue of having generators produce paraphrases in a preferred order is relevant. After all two of the main generation approaches (Systemic generation and Classification) use deterministic approaches and produce a single result. However, the issue of the large number of paraphrases that humans produce has not passed unnoticed in the literature. Here we reproduce couple of examples in support of the claim that a single semantics can indeed be expressed in numerous ways.

The first example is from [Mel'cuk 88]. The following sentence:

The Food and Drug Administration has seriously cautioned expectant mothers to avoid one of life's simple pleasures: a cup of coffee (*Newsweek*, Sep. 15, 1980).

The same sentence can be expressed also as:

Pregnant women have been earnestly warned by the FDA against drinking coffee, one of the small pleasures of life.

The underlying meaning of the sentence consists of roughly eight fragments, each of which is represented by a column in Table 7.1.

FDA	serious(ly) earnest(ly) stern(ly) strong(ly)	caution warn forewarn counsel put on guard address a caution issue a warning make public a caution a warning	pregnant women expectant mothers mothers-to-be during pregnancy while expecting a baby women ladies	to avoid to abstain from avoid should abstain should not not to
1	4	9	7	6

coffee drinking indulge in consuming coffee cup of coffee	one of something that is one of constitutes	small pleasures life's simple joys of life
6	3	8

Table 7.1: Periphrastic variants of sentences

Each column contains (nearly) synonymous expressions that convey the corresponding "piece" of meaning; the number under a column indicates the number of expressions or variants in it. Any variant from one column combines with almost any variant from another column, for instance:

The FDA has addressed a stern caution to ladies to the effect that while expecting a baby they should abstain from coffee, something that is one of life's small joys.

The FDA has made public a strong warning addressed to mothers-to-be: they should not indulge in consuming coffee, one of the simple pleasures of life.

This allows us to multiply the number of variants in each column:

$$1 \times 4 \times 9 \times 7 \times 6 \times 6 \times 3 \times 8 = 217,728 \text{ paraphrases}$$

True, some of these paraphrases will probably not satisfy selectional restrictions or other co-occurrence constraints. However, it is possible to think of even more variants for each semantic chunk. Thus we have quite a few paraphrases for the initial meaning.

Arguments making the same point are presented in [Ohmann 71, pages xxxi-xxxii]. "Put before 25 speakers a fairly simple drawing, ask them to describe in a sentence the situation it portrays, and they will easily come up with examples as:

1. *A bear is occupying a telephone booth, while a tourist impatiently waits in line.*

2. *A man who was driving along the road has stopped and is waiting impatiently for a grizzly bear to finish using the public phone.*

3. *A traveler waits impatiently as a bear chatters gaily in a highway phone booth.*

Almost certainly, each of the 25 sentences will be different from one another, yet each will adequately describe the drawing ... An analysis by computer shows that the 25 sentences about the bear in the telephone booth yield the material for 9.8 billion sentences, all describing just one situation."

So the point is that humans can come up with many paraphrases for a given meaning in a certain situation. The better the generation systems that we have the more of these sentences should be produced by generators. Yet, it is unlikely all of these sentences will satisfy our pragmatic goals equally well. Otherwise if they cannot be distinguished a natural question that arises is: why are they in the language in the first place? If some paraphrases are better than others and if the number of paraphrases is really large then it is interesting to consider techniques that will ensure that the better paraphrases will be produced first.

7.2 Choosing between paraphrases in generation

Several criteria can be used to judge alternative paraphrases:

1. semantic: semantic distance between the input semantics and corresponding semantics of the generated sentences

2. lexical: how parts of the semantics correspond to words (the semantic packaging of information); the distance between the semantics of individual lexical items and the part of the input semantics they 'cover', salience of the lexical items: dog vs. vertebrate vs. animal [Reiter 91], [Sowa 93].

3. syntactic: often there are alternative syntactic ways of expressing the same meaning. But some of these can be ambiguous, rather unwieldy, convoluted, unclear, etc.

4. stylistic: in different contexts that could mean using more (or less) varied syntactic structures; avoiding repetition; etc.

5. intonational: surface structures that have the same syntactic structures and the same corresponding semantics might be pronounced differently. Often different ways of pronouncing sentences yield different semantics but there are cases when they don't or it is not clear what these differences are.

In this chapter we will not look at the latter two issues—we just merely note them for completeness of the discussion. Obviously the above dimensions are interleaved and there are intricate interactions between them [Halliday 94].

7.3 Semantic aspects of choosing between paraphrases

In this section we focus on semantic aspects of choosing one paraphrase over another. We assume a generator that has a certain degree of flexibility in allowing the semantics of the different paraphrases to vary from the input semantics and also be different among themselves. We address how two semantic representations can be compared. Our proposal is specific to semantic representations cast in terms of conceptual graphs although most if not all of the framework applies to other representation formalisms as well.

In Section 7.3.1 we discuss possible approaches for finding the most similar semantic representation. We briefly review the existing work on comparing conceptual graphs in Section 7.3.2. Then we discuss the notion of what it is for a representation to be a better match to the input semantics in Section 7.3.3. We present our proposal in Section 7.3.4. This is followed by a discussion of some open issues (in Section 7.3.5) and an investigation of some properties of our relation (in Section 7.3.6).

7.3.1 Finding the most similar semantic representation

Given a representation for the input semantics the problem of finding the most similar semantic representation from the list of the semantic representations of the generated paraphrases can be approached at least in two ways:

1. Constructing a similarity (or distance) metric which can be used to determine the similarity between the input semantics and the semantics corresponding to the alternative paraphrases. Then (semantically) the best paraphrase will be the one whose semantics will have the maximum similarity (minimum distance) with the input semantics. We review similarity metrics for conceptual graphs in Section 7.3.2

2. Constructing a relation $<_G$ which holds between two semantic representations g_1 and g_2 ($g_1 <_G g_2$) iff the g_2 is a better match for the input semantics G than g_1.

It is desirable for the above relation $<_G$ to be:

1. transitive: $g_1 <_G g_2 \quad \& \quad g_2 <_G g_3 \Rightarrow g_1 <_G g_3$

2. irreflexive: $g_1 \not<_G g_1$

3. asymmetric: $g_1 <_G g_2 \Rightarrow g_2 \not<_G g_1$

The transitivity, irreflexivity and asymmetry properties make the relation $<_G$ a strict partial order. Naturally it would be nice to have the order be total (linear) but it is not always clear whether two semantic representations could be comparable in a principled way. Thus, the best match(es) can be defined as the maximal elements of the partial order.[1] All of what we have said about the relation $<_G$ stems from the nature of our task and not from an operational model that is assumed.

A similarity metric between the input semantics and the semantics of a paraphrase — $similarity(G, Built\_Sem)$ which gives a numerical value of how close the match between the two representations is, naturally suggests an ordering relation $\lhd_G$ defined as:

$$g_1 \lhd_G g_2 \text{ iff } similarity(G, g_1) < similarity(G, g_2)$$

The relation $\lhd_G$ holds if the semantic representation g_2 matches G better than g_1. If $similarity(G, g_1) = similarity(G, g_2)$ then whether $g_1 \lhd_G g_2$ is undefined. Thus, the

[1] A maximal element for a partial order $<_G$ is an x s.t. $\nexists y : x <_G y$

first approach is more restrictive (because it implies the second). We think that in the domain of semantic representations for an NL system it is difficult to construct a measure that will reflect the nuances of differences between two semantic representations. In a way, for the purposes of comparing the semantics of paraphrases, having such a measure is more than what is needed: what similarity (semantic distance) 5 or 7 means is not clear. We believe that it is easier to compare two semantic representations against the input semantics and come up with a judgement which one is a better match of the input semantics. This is exactly the nature of our proposal in Section 7.3.4. Before that we discuss existing approaches for comparing two conceptual graphs.

7.3.2 Existing approaches for comparing graphs

A number of techniques have been proposed in the conceptual graphs literature that deal with matching of conceptual graphs and we briefly review and evaluate them. Some of the techniques address how different graphs are (semantic distance measures), other techniques are concerned with how close the graphs are (similarity measures). It is straightforward to relate the two types of approaches by the following rule of thumb: The more the semantic distance between graphs is, the less similar they are.

The earliest attempts of defining the semantic distance between conceptual graphs are documented in [Garner *et al.* 87, Foo *et al.* 89]. The semantic distance between conceptual graphs is defined in the following way:

Given two concepts C_1 and C_2 the semantic distance between the concepts (sd_c) is:

$$sd_C(C_1, C_2) = \begin{cases} sd_C(C_1, C) + sd_C(C_2, C) & \text{where } C \text{ is a concept s.t.} \\ & type(C) = lub(type(C_1), type(C_2)) \\ \infty & \text{if } lub(type(C_1), type(C_2)) = \top \end{cases}$$

The semantic distance between two graphs G_1 and G_2 is defined as the sum of the semantic distance between corresponding pairs of concepts in the two graphs:

$$sd_G(G_1, G_2) = \sum sd_C(C_1, C_2)$$

where C_1 and C_2 are corresponding concepts in the maximal projection of G_1 and

G_2. If the graphs G_1 and G_2 are incompatible (cannot be joined) then $sd_G(G_1, G_2)$ is undefined. It is confusing that in the explanation of the metric in [Foo *et al.* 89] concepts whose types are sister nodes in the type lattice are given a distance 1 rather than 2. This approach only considers the concepts in the maximal projection. The amount of extra material in each graph is ignored.

Delugach addresses the notion of semantic distance in a more abstract (and less operational) way [Delugach 92]. He provides links from psychological studies and notions in philosophy. He notes that in practice the comparison of two objects is dependent on the order in which they are presented. Thus, $similarity(G1, G2)$ need not be the same as $similarity(G2, G1)$. Furthermore, whether one is interested in the similarity or difference between objects is likely to bias the results to the extent that the similarity cannot be seen as the opposite of the difference. The importance of context is stressed and some schemes are suggested that include taking into consideration the neighbouring concepts and assigning weights to concepts.

Yang et al. discuss the similarity between two CGs in the context of retrieving information [Yang *et al.* 92]. A query graph is matched against a database of previously stored graphs. The user can specify the degree of matching (in percentages) and thus filter out graphs that have less in common with the query. The similarity between two CGs is defined in terms of their common nodes:

$$Degree\ of\ matching\ = \frac{Number\ of\ matching\ concepts}{Number\ of\ concepts\ in\ query\ graph}$$

Another parameter of the matching procedure is the degree of inheritance which is a numerical value specifying the number of levels the types of corresponding concepts in the retrieved graph could differ. This restricts the retrieved graph to be either a generalisation or a specialisation of the query graph—situations where the retrieved graph generalises one concept but specialises another are disallowed. The above approach is computationally efficient but it fails to make an important distinction between retrieved graphs which introduce a different amount of extra material to the query graph.

An implementation of a weighted technique is presented in [Puder *et al.* 95]. In this work, which regards an open distributed environment as service market, a conceptual graph representing a requested service is matched against other conceptual graphs that

represent offered services. The main idea is that concepts and relations are assigned weights and the quality of the match between graphs is characterised by a value computed on the basis of the weights. As is often the case with weight-techniques , coming up with a principled way of assigning weights is difficult. Here again increasing the size of one of the graphs has no influence on the similarity (this is even stipulated as requirement M2 of the matching model).

The best attempt at formalising the notion of closeness of two conceptual graphs in a general way is the proposal by Poole and Campbell. In [Poole & Campbell 95] an algorithm is presented that determines the similarity between two graphs based on a (user-definable) measure of information content called 'interest'. The 'interest' of a graph is defined as a function that returns a numeric value for the amount of useful information contained in the graph. The similarity between graphs reflects the shared information:

$$similarity(G_1, G_2) = \frac{interest(G)}{max(interest(G_1), interest(G_2))}$$

where G is the generalisation of G_1 and G_2 which maximises the *interest* function.

The fact that the *interest* function can be defined in different ways allows for high flexibility. In particular it is shown how by varying the *interest* function surface, structural and thematic similarities can be achieved. However, it is extremely difficult to have a good 'interest' function in our domain because the 'interest' of nodes is a static and monotonic notion (i.e., the graphs are considered in isolation from any other graph). It also hard to consider the graph G as we cannot always guarantee that it will be between the lower and upper semantic constraints that we impose on corresponding semantics of valid derivations.

7.3.3 Minimal requirements: the intuitive notion of similarity

In order to justify our design decisions we look at some desirable properties of what it is to be a better match to the input semantics.

Informally a graph $G1$ will be considered a better match of the input semantics $IniSem$ than another graph $G2$ if $G1$ misses less detail from $IniSem$ (covers more of it) and

introduces less information than $G2$.

Obviously if we have generalisation chains: $G1 \leq G2 \leq IniSem$ or $IniSem \leq G2 \leq G1$ then $G2$ is a better match than $G1$. The interesting cases occur when $G1$ and $G2$ cannot be compared on the generalisation partial order.

Methodologically we will look at individual (local) aspects of difference between graphs (particular differences provided all other things are equal) and then try to combine individual differences to get the overall comparison.

Cases of deviation from the input semantics will include:

- generalising:
  - generalising the type of a concept or relation;
  - omitting a concept or relation;
- specialising:
  - specialising the type of a concept or relation;
  - introducing an additional concept or relation.

The above assumptions differ from previous research in that we compare two graphs against each other with respect to a third graph. Thus it is possible to have $cg_1 <_G cg_2$ and $cg_2 <_{G'} cg_1$ for different (reference) conceptual graphs G and G'. Also we do not construct a numerical measure but a relation.

7.3.4 Our proposal

We impose an important condition on the semantics of the generated paraphrases—namely that they be different ($Built\_Sem_i \neq Built\_Sem_j$ for $i \neq j$). This is not a restrictive assumption because if two paraphrases have the same semantics, which paraphrase is a better rendition of the input semantics will be judged according to the other criteria mentioned in Section 7.2.

Let $IniSem$ be the initial semantics; G_1 and G_2—the semantic representations corresponding to two paraphrases.

Definition 7.28 (Semantically closer to IniSem graph)

$$G_1 <_{IniSem} G_2 \quad \textit{iff} \quad S = S_1 + S_{2\_3} + S_4 + S_{5\_6} > 0$$

where S_1, S_{2_3}, S_4, S_{5_6} (which correspond to partial comparisons between G_1 and G_2 in the labelled areas in Figure 7.1) are defined as follows:

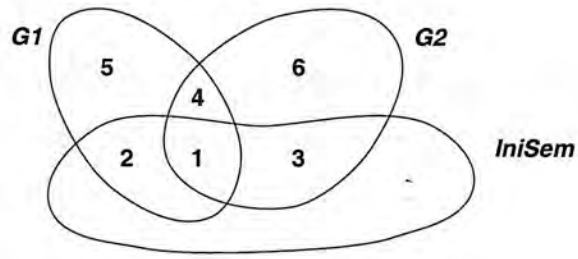


Figure 7.1: Comparing two graphs against the initial semantics

$\boldsymbol{S_1}$ Let C_1 and C_2 be corresponding concepts in

$$concepts(max\_proj(IniSem, G_1)) \cap concepts(max\_proj(IniSem, G_2))$$

(see area 1 in Figure 7.1).

$$S_1 = \sum weight(C_{IniSem}) \times (dist(C_{IniSem}, C_1) - dist(C_{IniSem}, C_2))$$

where the weight of a concept in the input semantics—$weight(C)$ is meant to reflect the importance of this concept in conveying the message. There could only be two kinds of concepts—obligatory to be expressed and optional (and thus only two weights).

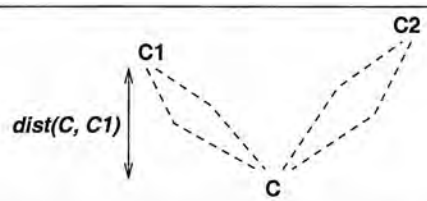


Figure 7.2: Distance between corresponding concepts

The function *dist* is a measure for the distance between two corresponding concepts:

$dist(C_{IniSem}, C_i) = k \times$ (the difference in levels between $type(C_{IniSem})$ and $type(C_i)$)

where

$$k = \begin{cases} \frac{penalty\ for\ specialisation}{penalty\ for\ generalisation} & \text{if } type(C_i) \leq type(C_{IniSem}) \\ 1 & \text{otherwise} \end{cases}$$

The difference in levels between two types is defined as the length of the shortest path(s) in the lattice between them (see Figure 7.2—C_i can be either a generalisation or a specialisation of C).

S_{2_3} Additional individual coverings of the initial semantics by the two graphs (areas 2 and 3 in Figure 7.1):

$$S_{2\_3} = \sum weight(C_{IniSem})\ dist(C_{IniSem}, C_1) - \sum weight(C_{IniSem})\ dist(C_{IniSem}, C_2)$$

S_4 Common information added (area 4 in Figure 7.1):

Let C_1 and C_2 are corresponding concepts in $concepts(max\_proj(G_1, G_2))$ but not in

$$concepts(max\_proj(IniSem, G_1)) \cap concepts(max\_proj(IniSem, G_2)).$$

Let C_L be the corresponding concept to C_1 and C_2 in the lower semantic bound (*LowerSem*).

$$S_4 = \sum (dist(C_L, C_2) - dist(C_L, C_1))$$

The idea is that the concepts in graph G_2 should be more general than their counterparts in G_1.

S_{5_6} Extra material added individually by G_1 and G_2 (areas 5 and 6 in Figure 7.1):

S_{5_6} = the number of concepts in the subgraph of G_1 in area 5 − the number of concepts in the subgraph of G_2 in area 6

S_{5_6} is crude. It can be refined using the counterpart concepts in the lower semantic bound. If all S_1, S_{2_3}, S_4, S_{5_6} are positive then G_2 matches better the input semantics G than G_1. The sum S will then also be positive. However, if the overall sum S is positive little can be said about the individual sums. They can be given different weights as in:

$$4S_1 + 3S_{2\_3} + 2S_4 + S_{5\_6}$$

7.3.5 Open questions

We have defined one approach for comparing two graphs against a third. In comparing two concepts we considered only their types. More sophisticated schemes will need to consider the referent field as well. None of the existing approaches addresses that. Yet we often find conceptual graphs in the referent field of PROPOSITIONs. We can try and define $dist($ [PROPOSITION: G1], [PROPOSITION: G2] $) = dist(G1, G2)$. This would require having a function $dist : CG \times CG \rightarrow real$ which we were trying to avoid. One way to go about this problem is to stipulate that $dist(G1, G2) = S$ where S is the sum we defined earlier.

The scheme we presented above requires that the input semantics is annotated with what concepts are obligatory and what concepts are optional. In addition in order to calculate the sum S we would need to have the weights for obligatory and optional concepts. At the moment we do not have a methodology for coming up with these weights in a principled way. In fact one need not consider a black and white distinction only. There can be more than two types of 'obligatoriness' of expression. Notions like salience also become relevant. Likewise the ratio of the penalties for specialisation and generalisation is a parameter that requires a lot of experiments. We use a number less than one because intuitively if we introduce more detail than specified in the input semantics this seems to be less of a problem than missing information (generalising).[2]

[2] Some people are known to introduce more detail without this being necessary contributing to the main point.

7.3.6 Properties of our relation

In this section we make some simple observations regarding the relation $<_G$ defined in Section 7.3.4.

Irreflexive: $CG <_G CG$ is false for all conceptual graphs CG.

Proof: Because G_1 and G_2 are the same we only have the sums S_1 and S_4 which are both 0. Thus $S = 0$ and the condition in Definition 28 does not hold. □

Asymmetric: $G_1 <_G G_2$ implies that $G_2 <_G G_1$ is false.

Proof: From $G_1 <_G G_2$ by Definition 28, it follows that $S > 0$. If we considering $G_2 <_G G_1$ the sum that we would have to form will turn out to be $-S$ which is negative. Thus $G_2 <_G G_1$ cannot hold. □

Greatest element: The graph G is the greatest element for the relation $<_G$. Given a graph G and a graph G_1, the relation $G_1 <_G G$ always holds.

Proof: The negative elements in the sums S_1 and S_{2_3} will be zero—thus the sums will be positive. The sum S_4 will not be formed because there would not be area 4. Area 6 would not exist, hence S_{5_6} will be positive. Thus, the overall sum S will be positive. □

Specialisation/Generalisation chains: Given three graphs G_1, G_2 and G which form a specialisation (or generalisation) chain—$G_1 < G_2 < G$ (or $G_1 > G_2 > G$) the following relation holds: $G_1 <_G G_2$

Proof: $S_1 > 0$ because the distance between corresponding concepts in G_1 and G is greater than for G_2 and G. S_{2_3} would not exist because G_2 is properly contained in G_1 (see Figure 7.3). S_4 will be positive because G_2 is a generalisation of G_1. S_{5_6} is positive because the counterpart of area 6 doesn't exist. Thus, $S = S_1 + S_{2\_3} + S_4 + S_{5\_6} > 0$ and $G_1 <_G G_2$. □

On transitivity: the relation $<_G$ is not in general transitive. From $G_1 <_G G_2$ it follows that the corresponding sum S is positive:

$$S = S_1 + S_{2\_3} + S_4 + S_{5\_6} > 0$$

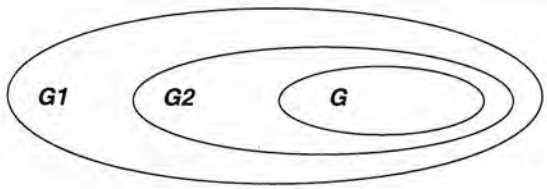


Figure 7.3: $G_1 <_G G_2$ when $G_1 < G_2 < G$

From $G_2 <_G G_3$ follows that a similar sum S' is positive:

$$S' = S'_1 + S'_{2\_3} + S'_4 + S'_{5\_6} > 0$$

The above two statements do not allow us to infer that the corresponding sum of graphs G_1 and G_3 (S'') is positive too. We might want to consider a definition for the sum S in Definition 28 that requires all subsums S_1, S_{2_3}, S_4 and S_{5_6} to be positive. Even with such a stronger definition we cannot state that for the individual subsums the following inequations hold: $S_1 < S'_1 < S''_1$, etc. (S''_1 being the corresponding subsum for G_1 and G_3) because the sums in general can be formed over different corresponding concepts.

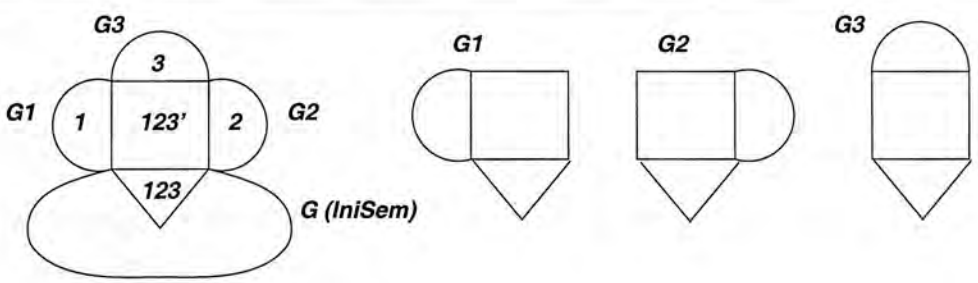


Figure 7.4: $G1 <_G G2 <_G G3$

There is a special case when the relation $<_G$ is transitive. This is when all the graphs cover exactly the same portion from the graph G (area 123 in Figure 7.4) and when they all share exactly the same common parts (outside of G)—area 123′ in Figure 7.4. In that case the relation is a strict (strong) order.

The relation $<_G$ defined in Section 7.3.4 is not in general a partial strict order. It is irreflexive and asymmetric yet not always transitive.

7.4 A sketch for syntactic preference

In this section we turn to syntactic preferences. We note that among paraphrases that differ syntactically it might be desirable that some paraphrases are generated earlier than others. We briefly mention a couple of grammatical constructions which though grammatical are not recommended if there are other ways to express the same meaning. Our purpose is not to be exhaustive,[3] but to motivate that mechanisms for comparing paraphrases syntactically are needed in preference-based generation.

Heavy NPs: There is no exact definition of what heavy NPs are. Informally these are NPs with a lot of lexical material (i.e., quite long). The reason why they matter is they can influence the preferences among constructions and even worse rule out the acceptability of some constructions which are perfectly normal with small NPs. Heaviness plays an important role in modifier attachment as shown in Table 7.2.

John sold it today.	* John sold today it.
John sold the newspapers today.	?/* John sold today the newspapers.
John sold his rusty socket-wrench set today.	John sold today his rusty socket-wrench set.
? John sold his collection of old newspapers from before the Civil War today.	John sold today his collection of old newspapers from before the Civil War.

Table 7.2: Heaviness and word order

Light adverbials (e.g., single word adverbials) tend to appear before heavy arguments (and likewise heavy adverbials tend to appear after them).

It's not a good idea for heavy NPs to split verb-particle constructions:

(7.1) *I rang her up*
(7.2) *I rang my friend from Edinburgh up*
(7.3) ? *I rang my friend who comes from Edinburgh up*
(7.4) * *I rang the girl I met at that party in East Slope where Mike was making such a fool of himself over that gigly blond in the fishnet stockings up*
(7.5) * *I rang up her*

Center embedding: These are constructions which have a lot of nesting of constituents in the middle of the sentence (cf. Figure 7.5). Sentence 7.6 is a simple example

[3] After all, giving prescriptive guidelines as to what constructions are preferred (in different circumstances) is the field of (Second) Language Learning (2LL).

which can be paraphrased using the declarative transitive construction with a relative clause (Sentence 7.7):

(7.6) *The dog the cat the mouse ate chased.*
(7.7) *The dog chased the cat which ate the mouse.*

Sentence 7.8 is a more complicated (artificially constructed) example (its syntactic structure is presented in Figure 7.5). 7.8 is fairly hard to understand (if at all). Even its paraphrase 7.9 is hard to comprehend.

(7.8) *The book the professor the students who are doing well like recommended is good.*
(7.9) *The book that was recommended by the professor whom the students that are doing well like is good.*

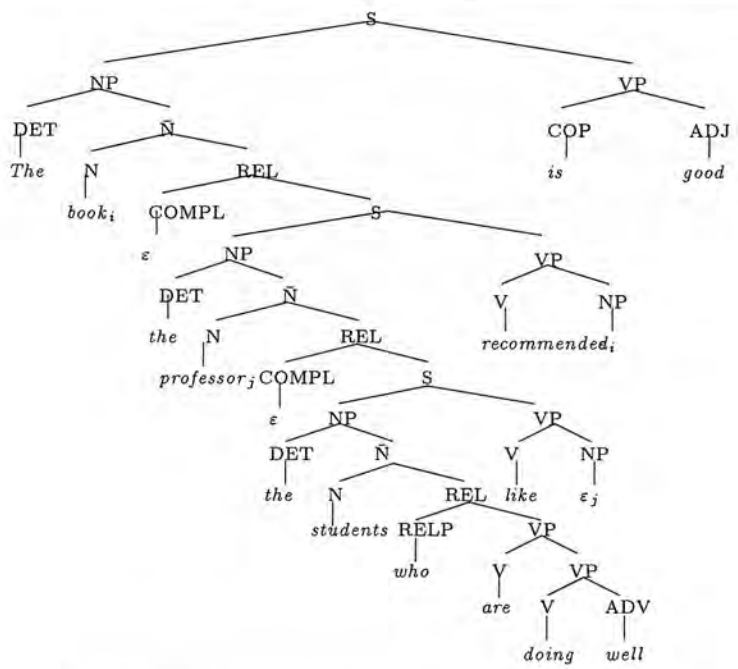


Figure 7.5: Syntactic structure for sentence 7.8

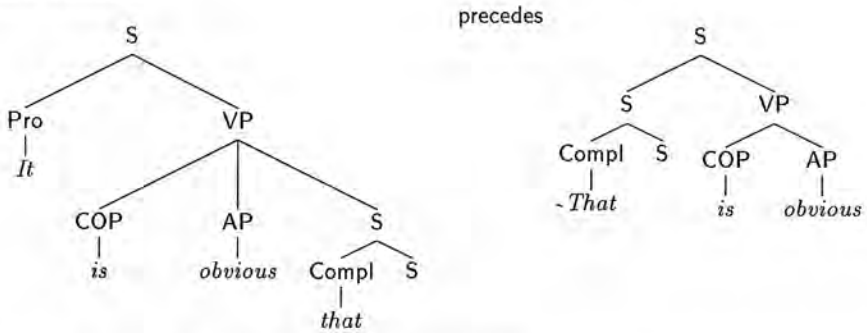
Here is an actual naturally occurring example (we use different fonts to mark the different clauses):

(7.10) *The money which the members of staff who take coffee in the coffee room placed in the saucer on Monday has been stolen.*

Sentential subjects: These are sentences in which the subject is itsself a sentence.

(7.11) *That Nixon is a crook is obvious.*
(7.12) *It is obvious that Nixon is a crook.*

The second construction (it-extraposition) is the more natural one. In fact the first construction is only used by people that have had higher education (Larry Trask p.c.). Such preferences can be achieved by imposing a linear order on the mapping rules for the constructions:



Split infinitives: This is a grammatical construction in which the particle *to* and verbal part of an infinitive (e.g., *love*) are split by an adverbial phrase (*to madly love*, *to really and truly love*) or other word(s). Here is a famous example:

(7.13) *to boldly split infinitives where no man has split before*

There is a noticeable reluctance to split infinitives in written English [Burchfield 96, page 737]:

(7.14) *... there will be a further disposition* ***seriously to underestimate*** *the strength ... of the United States.*

(7.15) *He was never ashamed* ***publicly to bear*** *witness.*

How can we account for these preferences? One approach is to impose an ordering on the sister-adjunction constraints (SACs, see Section 4.3.6). So for VP_2 we could have the sister-adjunction constraint (right,AdvP) precede (left,AdvP). Yet, if the NP is heavy then a post modifier for VP_2 is not possible (and in some cases it might not be possible to avoid split infinitives in order to avoid ambiguity):

(7.16) *She decided* ***to gradually get rid of*** *the teddy bears she has been collecting over the last twenty or so years.*

She decided ***gradually to get rid of*** *the teddy bears* is ambiguous (it is not clear whether the decision was gradual or the getting rid of the teddy bears). Thus, due to

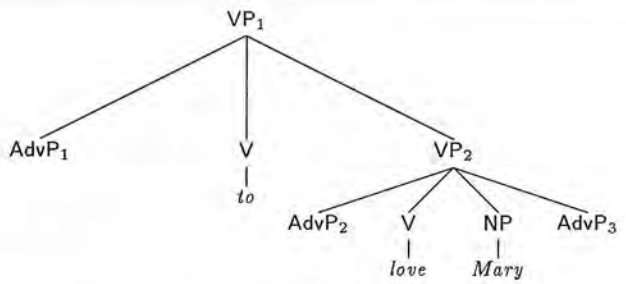


Figure 7.6: Fraction of a d-tree with adverbial modifier positions

the incremental nature of the way we handle modification the notion of heaviness of an NP might require a re-ordering of structures if we add material to them. We will return to these issues in the discussion (Section 7.6).

7.5 Best-first generation: incorporating preferences

In this section we look at ways of incorporating the ideas of preferring (partial) structures on the basis of the semantic, and syntactic criteria we have discussed above.

Firstly we need a more complicated generation strategy that would allow for specification of the preferences. In chart parsing a common technique for implementing flexible control is to introduce a data structure called an *agenda* which will contain the "pending" edges that we would like to add to the chart. Instead of putting an edge in the chart when we have constructed it (by completing) or determined that it is useful (in the prediction step), we add it to the agenda. By imposing a certain discipline on how edges are added to the agenda and how they are removed from the agenda we can get different flavours of processing.[4] We can use the same idea in generation to gain more control over the order in which edges are entered into the chart and implement the preferences as considering 'better' edges earlier.

[4] For a comprehensive discussion of the use of an agenda in parsing see [Gazdar & Mellish 89, page 206ff].

7.5.1 Agenda-based control

In this subsection we talk about a generation strategy which uses an agenda. We will consider a top-down strategy though it should be obvious that other strategies are easily implementable, more so than in other places in this thesis.

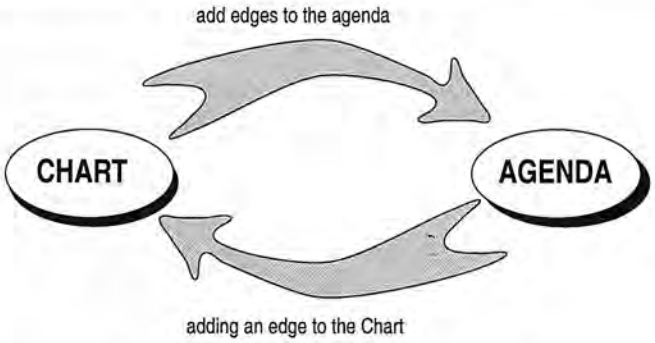


Figure 7.7: Agenda-based control

Generation starts off by initializing the agenda. We can consider what possible mapping rules match the input semantics and create the corresponding edges. Instead of putting them directly in the chart we can put them on the agenda.[5] Then we can take an edge whose syntactic structure is rooted at S and put it in the chart. Having a new edge in the chart might result in possible interactions with other edges in the chart (due to the fundamental rule) or we can predict active edges corresponding to the generation goals of the original edge. We gather all such new edges and again instead of putting them immediately in the chart we add them to the agenda. We will assume the agenda maintains a reasonable order of the edges that are there that corresponds to our notion of 'betterness'. Thus we want to take the next best edge to add to the chart. Taking edges from the agenda and putting them in the chart results in new edges that are created and that are added to the agenda. Thus we get the cyclic process in Figure 7.7.

The invariants at the end of each cycle are:

- none of the interactions between an edge on the agenda and other edges have been explored.

[5] Immediately after this chart parsers with an agenda need to consider initial top-down prediction edges $\langle 0, 0, S \rightarrow \bullet\ \alpha \rangle$. We don't need to worry about these as we use a lexicalised grammar.

- all possible interactions between an edge in the chart and other edges in the chart have been reflected either as edges in the chart or on the agenda.

We have stated more precisely the top-level algorithm for chart generation with an agenda in Figure 7.8

```
PROCEDURE chart_gen_with_agenda(Sem : cg_graph);
global:
    Chart: look-up table indexed on semantics;
    Agenda: sequence of priority queues;
begin
    Chart := empty;
    Agenda := empty;
    initialize_agenda(Sem, Agenda);
    start_active(Sem, Agenda);
    until Agenda is empty do
        remove_edge(Edge, Agenda);
        put_edge_in_chart(Edge, Chart);
        new_edges(Edge, Chart, NewEdges);
        add_edges_to_agenda(NewEdges, Agenda);
    announce_result(Chart);
end
```

Figure 7.8: Algorithm for chart generation with an agenda

The procedure *initialize_agenda* produces an initial *Agenda*. In CFG parsing the initial agenda will contain inactive lexical edges for every input word.[6] In DTG generation we have considered three approaches:

1. We could select all mapping rules that match the input semantics *InputSem*. Because our grammar is lexicalised we can have mapping rules that have internal generation goals or mapping rules that don't. Then the remainder of the generation process is trying to figure out how these can be put together.

2. One can split the lexicalised grammar into a collection of trees and words and in the initialisation stage find the words and then worry about what trees can be anchored by what words.

[6] In fact in top-down chart parsing with an agenda these edges are directly put in the chart as they lead to no interactions between themselves.

3. The initial agenda can be kept empty. Later if lexical items are predicted they can be looked up in the word databases (assuming a separation of the trees and lexical items). This is not as bad as blind top-down parsing because at every stage mapping rules are going to be licenced by a match with the semantics.

The procedure *start_active* creates the initial active edges that will get the process going. For every mapping rule that matches the input semantics and whose syntactic structure is rooted at the distinguished symbol (usually S) add a corresponding edge to the agenda (initial top-down prediction).

The procedure *remove_edge* takes the next 'best' edge from the *Agenda* (i.e., the *Agenda*now contains one edge less). We will discuss the agenda at length in the next section. For now we want to get a quick overall idea of how things work at a more abstract level.

The procedure *put_edge_in_chart* puts the edge in the semantically indexed chart.

The crucial thing happens when procedure *new_edges* considers the interaction between the newly added *Edge* and the other edges in the *Chart*. New edges that result from possible interactions are in *NewEdges*.

Finally, procedure *add_edges_to_agenda* adds the *NewEdges* created in the previous procedure to the *Agenda*.

The process continues until there are no more edges in the *Agenda*.

At the end we look in the chart to see if we have a well-formed sentence that we have generated. This is the job of the procedure *announce_result*. We look for edges whose syntactic structure is rooted at an S and that do not have unexplored generation goals and have consumed the input semantics (or have a best approximate semantics match to the input semantics). Sometimes the above model of reporting the successful derivation is not very good (especially when time is of essence). The reason for that is that we do all the work and only then report the generated sentences. Some of these sentences are generated quite early on in the process. The best sentences are created first (except that some incomplete variants of the best sentence necessarily come before that). One way to address this issue is to put a check on the procedure *put_edge_in_chart* that

would verify if the edge is an adequate solution. Here the expensive part is checking that the semantics of the generated sentence is close to the input semantics. However, if we were interested in exact generation we could map (partial) semantic structure onto bit vectors and use these to determine whether a semantic structure is the same as the input semantics. This operation is rather quick.

So far, so good, except we have been rather general about what adding an edge to the *Agenda* and taking the next 'best' edge mean. We do this in the next subsection.

7.5.2 The structure of the agenda

Suppose, for the sake of concreteness, we have one initial top-down prediction for a sentence which we have taken from the agenda and have put in the chart. Let this edge have three internal generation goals (e.g., a ditransitive construction) and let's consider what happens halfway through adding edges to the agenda for the last complement. The observation we want to make is that there is no point whatsoever comparing this edge to any of the edges for the other complements. Because the issue of whether it should precede them or not depends on which complement we would like to expand first and not on how well this particular edge compares to edges for other complements. This is because the other edges have a different semantic head. So we want to compare edges with the same semantic head only. We want to have some structure in the agenda—each individual section should be indexed on different semantic concepts.

Within an individual section it might happen that we can have edges with different top level categories (for DTGs maximal projection nodes, i.e., the nodes that can be substituted in a subsertion operation; see Section 4.3.2). For example we might want to express a complicated piece of semantics as an S or a complex nominalisation phrase—an NP. Again comparing an edge for an S and one for an NP makes no sense as they are not competing between themselves (for any generated sentence we could not replace one with the other).

Finally, the edges that we will want to compare we can put in a priority queue.

If we have a new edge we have to find its queue first (indexed on the semantic head and the syntactic category) and then add it to the priority queue. Adding an edge to

a queue could result in it being not just at the end of the queue but possibly further to the front if it has a higher priority (i.e., is a 'better' edge than some of the existing ones). That is why we need a priority queue.

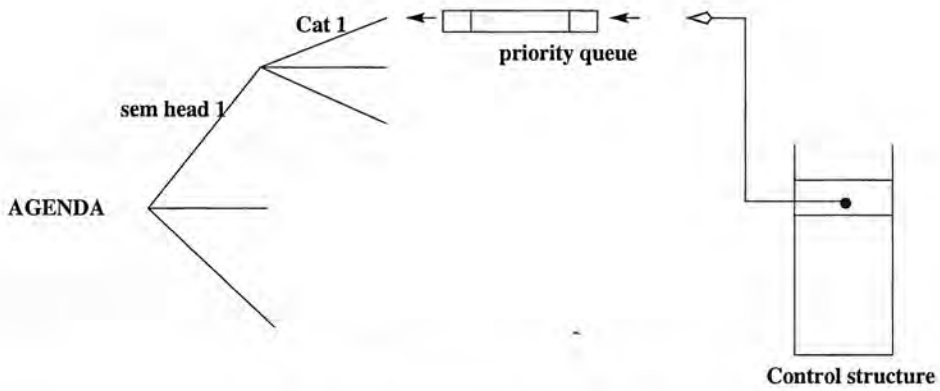


Figure 7.9: Structure of the agenda

Now let's consider how we can get edges out of the agenda. In the case for parsing if we wanted to have top-down strategy (depth-first search) we would have a stack for the agenda. As we discussed above we are tied to such a structure for the agenda due to the nature of the generation task. Yet, we can still have a top-down strategy if we kept a separate *control structure* that would tell us from which section to take the next 'best' edge. This control structure is organised as a stack of elements consisting of the primary and secondary keys (semantic head and syntactic category) of edges. The important thing to note is that the control structure does not specify which edge to pull but which priority queue to consider next. The control structure is built according to how elements are added to the chart (assuming top-down strategy). When we take edges from the agenda it is not the same edges that we take but the 'best' edge in the particular queue. The number of elements in the control stack is the same as the number of edges on the agenda.

The organisation of the agenda is fairly intuitive, yet considerably more complicated than any other proposals in the literature.

7.5.3 Notes on implementation: priority queues and heaps

In the organisation of the agenda we use a stack (for the control structure) and priority queues for individual subsections (indexed on semantic head and then on category of the maximal projection node).

A **stack** is a data structure which can be thought of like a pile where it is convenient to remove the top object on the pile or add a new one above the top. Stacks are well described in the literature (e.g., [Aho *et al.* 83, page 53ff]).

A **priority queue** is data structure for maintaining a set of elements, each with an associated value called a key. A priority queue supports the following operations:

CREATE(PQ) creates an empty priority queue PQ.

INSERT(PQ, e) inserts the element e in the priority queue (set) PQ.

MAXIMUM(PQ) returns the element of S with the largest key.

EXTRACT_MAX(PQ) removes and returns the element of PQ with the largest key.

We use a heap to implement a priority queue. A heap is a well-known data structure and good descriptions of heaps and how they can be used to implement priority queues can be found in [Cormen *et al.* 90, page 149] [Helman & Veroff 86, page 468] [Manber 89, page 68]. A heap can be thought of as a binary tree whose nodes contain an element and a key. The key of each node is greater than or equal to the key of any of its daughters. Heaps are a very good data structure to implement priority queues because they support inserting and removing an element from a priority queue on a set of size n in logarithmic time ($O(lg\, n)$).

7.6 Discussion

Our initial starting point for semantic comparison was the approach by Nogier and Zock in which they count the number of concepts in the maximal projection when they match (maximally join) the input semantics with the semantics of lexical items [Nogier & Zock 92]. In that approach there is no separate knowledge base which reflects the state of affairs and a maximal join which results in a specialisation of the input

semantics is not allowed.[7]

In the context of MT a preference approach is considered in [Wu & Palmer 94]. Wu and Palmer's similarity measure allows for correct lexical choice to be achieved even when there is no exact lexical match (semantically) from the source language to the target language.

The CORE LANGUAGE ENGINE [Alshawi 92] system also considers rating the paraphrases generated with the semantic head-driven algorithm. The method used there is quite different. It is based on statistical preferences. Simple word n-gram models are used to select between alternative sentences generated from the same quasi-logical form (QLF). The overall probabilities are taken, then they are normalised for length, and the sentence with the highest score is choosen. On a limited domain the initial experiments that have been conducted worked surprisingly well (Stephen Pulman, pc.). Use of statistical biases to select from alternatives in generation is reported in [Knight & Hatzivassiloglou 95]. Such statistical approaches naturally allow for handling adjacent collocations without explicitly representing such knowledge in the generator.[8]

Comparing representations and finding a best match is a recurring problem in AI and similar scenarios (to mention just a few) include:

- CASE-based reasoning: given a case base of previous situations, retrieve the case most similar to a new situation [Kolodner 93];
- Classifying new objects: given a hierarchical structuring of the domain of objects and a new object, find the most specific class that the object is an instance of [Reiter & Mellish 92].

In natural language generation, however, the input semantics is new and so is the list

[7] See also [Kukich 88] for fluency problems in NLG.

[8] Rather worryingly the authors advocate statistical filtering as an opportunity to simplify the grammar allowing it to overgenerate (e.g., get agreement wrong). They argue that they aparent ingrammaticallity disappears after the statistical filter is applied. We strongly appose this view. Encoding linguistic generalisations is what NLG (and NLA) is all about. Replacing a linguistic constraint which is easily represented and cuts down on the size of the search space with what in effect is a black box is not a good idea. The symbolic and statistical approaches should be used in a complementary fashion.

of generated sentences with their corresponding semantics. Thus, it is not possible to resort to complicated indexing mechanisms to preprocess (off-line) the list of elements among which we are searching as is standardly done in information retrieval.

It might seem that if at every level of the search space we choose the best available alternative (local, hill climbing, best-first search) we might reach the optimal paraphrase. Unfortunately, as we saw in the syntactic preferences concerning heavy noun phrases, we might start off a sentence with a better construction and then as we gradually add material to an NP it might become heavy enough so as to make a previous choice (say of a sentential construction) not be optimal.

The preference-based approach is computationally not cheap. Can we devise techniques that would allow us to perform some kind of off-line precompilation? Compilation for NLG has been discussed in [Patten *et al.* 92] but in a very different setting.

7.6.1 Alternative to semantic closeness: Classifying the input

7.6.1.1 Hierarchy of mapping rules

In this section we examine the ramifications of using the generalisation hierarchy of the semantic part of the mapping rules. In order to have the generator produce more appropriate sentences earlier we consider using a generalisation hierarchy of the semantic structures of the relation *semantics→syntax* building on work on the use of classification in generation [Mellish 91].

The semantics in the mapping rules (which we consider to be CGs) define a generalisation hierarchy. We take this hierarchy and instead of having only a CG at each node of the lattice we have a mapping rule. We call this structure a hierarchy of mapping rules and it can be computed off-line. In fact compiled hierarchical retrieval of conceptual graphs is one of the strengths of the semantic formalism that we use [Ellis 95]. The generation algorithm will be different from the current algorithm used in e.g., IDAS (because of the way we handle modification).[9] Such an approach can be seen as a special instance of classification generation where the semantic structures

[9] Going in depth in the precise details of what the generation algorithm could be is beyond the scope of the discussion here.

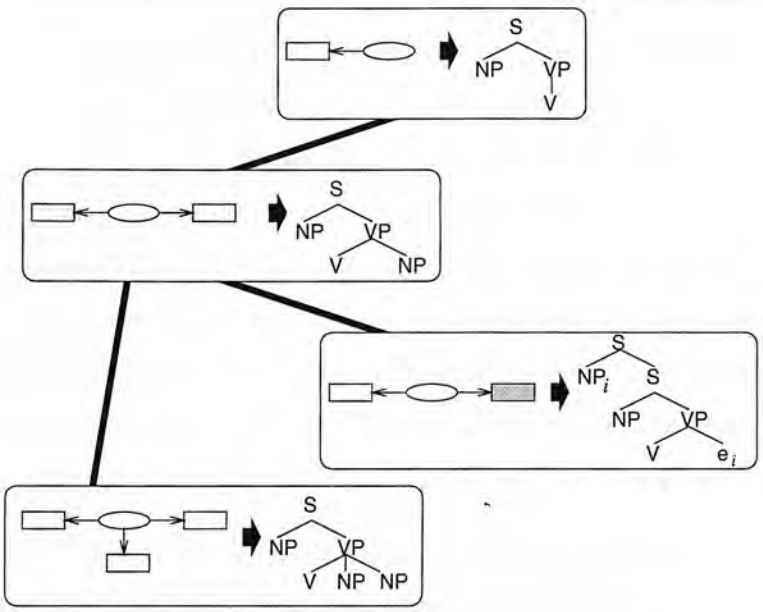


Figure 7.10: Hierarchy of mapping rules

are more complex than the I1 classes and likewise the syntactic structures, on their side, are more complex than the CFG rules that have been used so far in classification generation.

Again there are a lot of parallels between preference-based NLG and parsing. Preference-based processing for parsing is an area that has occupied psycholinguists a lot. More recently there has been a revived interest in statistical parsing which is again an instance of a preference-based approach. Other areas of AI which are based on preference processing include approximate reasoning and fuzzy systems, and some of the mechanism used in them might as well be applicable to preference-based NLG.

7.7 Conclusion

In this chapter we studied preference techniques for generation. We motivated the need for preference processing based on the fact that there are a lot of paraphrases for a given input. We looked at ways to compare generated structures from a semantic point of

view. We defined an approach which compared two conceptual graphs in the context of the input semantics (and the lower and upper semantic bounds). The mechanisms that we developed are more general and can be applied to other formalisms than conceptual graphs too. We also briefly looked at cases where syntactic preferences are needed. We showed an approach where preferences can be incorporated in the processing model by using an agenda which contains edges/structures that the generator is about to consider. The structure of the agenda is more complex than any existing proposals and we used an additional control structure to guide the use of the agenda. We related preference-based generation to the classification approach and to compiling syntactic preferences.

There are various ways in which the current work can be extended. The agenda-based control that we have defined for chart generators allows for different generation strategies to be related through a different way of exploring the agenda. This is similar to the way search techniques are presented only with a variation of the order unprocessed goals are considered using an agenda of the unprocessed goals. The generation framework that is built can be viewed as a general test-bed for experimenting with possible generation strategies.

SUMMARY

- In our model of semantic comparison we compare two conceptual graphs (applicability semantics of different mapping rules or the corresponding semantics of paraphrases) with respect to a third (input semantics).
- Syntactic comparison happens after semantic comparison.
- Agenda-based control allows for preference-based generation and for relating different generation strategies through a uniform processing mechanism.
- Generated structures can be evaluated and better structures can be processed earlier—this implements (local) preference-based generation.
- Best-first generation is useful when the search space is large and we cannot try all possibilities.

Chapter 8

Conclusions

In this chapter we revisit the original motivation for our research and examine to what extent we have accomplished the goals we have set ourselves. We discuss limitations of our approach and directions for future work (divided into short-term and long-term enhancements).

In sentence generation there are a number of standard assumptions that we believe need to be challenged. These are:

1. the hierarchical nature of the input to generators;
2. the exact specification of what should be conveyed.

In this thesis we have looked at the following aspects:

1. Use of non-hierarchical structures as input to generators. This allows us to investigate a more general version of the sentence generation problem where one is not pre-committed to a choice of the syntactically prominent elements in the initial semantics. Existing generation approaches often assume a hierarchical structure on the input semantic representation which is not justified independently of the language.
2. Use of 'upper' and 'lower' bounds on the semantic input which express the least and the most that should be conveyed. This allows for more flexibility in expression and mirrors better what happens in real text production by humans.

In addition we have also looked at:

3. Use of linguistically motivated syntactic structures. The previous studies in the conceptual graphs tradition (and for that matter some smaller scale generation systems) do not rely on proper linguistic theories and thus fail to capture generalisations which leads to problems when attempting to expand such systems.

4. Declarative specification of the relation between meaning and form. We use a declarative framework for encoding the mapping relation between semantic and syntactic structures.

5. Choice between alternative paraphrases on semantic and syntactic grounds. The standardly assumed constraints on the semantics of the output (completeness and coherence) are more restrictive than necessary and intuitively good realisations can be ruled out. We provide more flexible mechanisms.

6. Generation strategies and mechanisms for controlling the search. The generation task is in essence a search task. The search space in the context of generation is very big. In addition our assumption of non-hierarchical input makes the problem even worse. Some generators use a top-down backtracking strategy (e.g., FUF [Elhadad 93], SHDG [Shieber *et al.* 90]) and throw away all useful computation they might have performed in the course of exploring an unsuccessful branch of the search space.

7. Lexical choice is considered as an integral part of the surface generation problem. We do allow for subgraphs to be expressed by a single word (packaging of information).

We have addressed the above issues by developing a practical generation system with the following characteristics:

1. The input to the system is a conceptual graph, i.e., a non-hierarchical representation. The formalism of conceptual graphs stems from semantic networks. Conceptual graphs offer the advantage that the formalism has well defined deduction mechanisms.

2. The generator uses a notion of 'upper' and 'lower' semantic structures. The 'upper' semantics subsumes the input semantics which in turn subsumes the 'lower' semantics. If for some reason the generator is forced to further specialise the semantic type of an element in the input semantics the new semantic type should not be more specific than the type of the corresponding element in the 'lower' semantics. Conversely, if the generator generalises a concept from the input semantics the new concept should be more specific than the counterpart concept in the 'upper' semantics. Thus the semantics of the generated sentence (which we assume is not always identical to the input semantics) should be subsumed by the 'upper' semantics and it should subsume the 'lower' semantics. Such functionality is particularly useful if generation systems are to be used in real world applications where large knowledge bases (not necessarily created with linguistic purposes in mind) do exist and the sentence planning component relies on them to produce the semantic input for the surface realisation module. Thus, the links between the elements in the knowledge base and the elements in the input semantics can be established trivially.

3. The output of the generator is a linguistically motivated syntactic structure. We make use of the formalism of Lexicalised D-Tree Grammars which stem from work on Tree-Adjoining Grammars. Two important properties of D-Tree Grammars make them particularly suitable for generation:

   - Extended Domain of Locality: DTGs have a larger domain of locality than context-free grammars and linguistic formalisms based on them (such as Lexical Functional Grammars and Head-Driven Phrase Structure Grammars). Thus, the dependencies between a verb and all of its arguments can be stated (locally) in one construction.
   - There is a close match between the operations on the syntactic structures (subsertion-insertion and sister-adjunction) and the operations of complementation and modification on the semantic structures. This simplifies the mapping between syntax and semantics and also leads to a simpler generation architecture.

   By using a well-established syntactic theory and framework we have a system

which:

- captures language generalisations (parsimony);
- allows for new constructions to be added easily (extensibility); and
- is relatively simple to debug (maintenance).[1]

4. We have devised a formalism in which to encode in a declarative way the mapping between semantic structures expressed in terms of conceptual graphs and the (minimal) syntactic structures expressed in terms of (lexicalised) DTG trees. Such declarative mapping of conceptual graphs to linguistically motivated syntactic structures had not been done before. Because the DTG elementary structures have an Extended Domain of Locality within which dependencies are localised, the semantic counterparts of elementary syntactic structures are meaningful semantic units.

5. Generators often can produce a number of different paraphrases. We have developed a framework to compare which of the paraphrases will be closest to the initial semantics. Our generation model allows for paraphrases to express less (or more) information than specified in the input semantics. Such omissions (additions) should be minimal.

6. We have developed algorithms (generation strategies) which use declarative mapping rules and produce syntactic paraphrases for a given semantic input. In our generation model we use operations similar to subsertion-insertion and sister-adjunction. The standard DTG operations have been extended to take into account the combination of semantic structures. In order to control the search in the generation process we have developed special heuristics (indexing of rules, etc.). We have explicitly demonstrated the declarative nature of the mapping rules by exploring a number of different generation strategies: top-down generation strategy and a chart-based generation technique. The latter potentially allows for a whole array of generation methodologies to be explored using the same linguistic knowledge. The main advantage of using the chart technique is

[1] The compiler for mapping rules provides syntactic sugaring which hides unnecesary detail from the developer.

to avoid re-computing useful results which have been generated in the course of exploring an unsuccessful branch of the search space.

7. Our generator does not assume that its input already contains information about which content words will be used. Lexical choice is performed by the generator.

By combining two powerful frameworks - that of Conceptual Graphs and D-Tree Grammars, we have developed a generation framework which allows for the linguistic realisation problem to be viewed in a more general setting. In developing the syntactic part of mapping rules we have followed the TAG analyses of the XTAG system [Doran *et al.* 94]. New mapping rules can further be added to the generator. The generation system is implemented in LIFE and has a graphical tree displayer and a character-based interface for interacting with it.

8.1 Evaluation

NLG systems are hard to evaluate because:

1. they have different starting assumptions (for example what the input should contain and in what form);

2. they have different goals regarding what output they produce and how this output is used;

3. they use different architectures for managing the resources, different implementation platforms, etc.

We have tested PROTECTOR on a corpus of a few hundred examples and the system is robust and very quick (cf. the trace we showed in Section 6.2.5). We intend to do the following further experiments with our system:

1. showing the variety of syntactic constructions that the system can generate;

2. for a given input showing the paraphrases that the system can produce;

3. comparing non-memoization and memoization techniques:

(a) Calculating overhead of memoization for sentences that do not involve backtracking;

(b) Calculating speed-up for sentences that involve backtracking (e.g., sentences with lexical gaps);

(c) Calculating speed-up for generation of alternative paraphrases.

8.2 Limitations

Our approach is in essence a simple mapping from semantic structures to syntactic ones. While such an approach has advantages it also has limitations when the fuller range of tasks to be performed in a generation system (in the broad sense) are considered. In this section we discuss limitations of our approach some of which can be readily addressed and we touch upon how in the section on further work.

1. mapping rules are not distinguished on the basis of functional information (i.e., *differences in communicative effects* are not accounted for). Only the content of the utterance is given as input (although salience of concepts plays a role in the semantic comparison);

2. lack of heuristics for choosing/preferring an initial syntactic head among the set of possible ones;

3. the listener is not taken into account;

4. no account for context information;

5. in our system we have looked more at lexical packaging and less so at quantificational structures;

6. collocations are not accounted for. The use of a lexicalised grammar (DTG) and the fact that we generate syntactic heads first puts us in a good position to treat collocations (most collocations are relations between the syntactic head and the complements/modifiers). This requires more sophistication in the process of choosing lexical items;

7. in this work we are not addressing tense and aspect issues.

8.3 Further work

This section is devoted to possible extensions of the work. The system is viewed as a general test-bed for experimenting with possible generation strategies. Taking additional knowledge sources into account during generation is discussed.

8.3.1 Short-term improvements

The new sister-adjunction: Recent changes to the DTG formalism involve changing the sister-adjunction operation and the modification trees so that the modification tree projects up to the category that it modifies. The new sister-adjunction operation merges (unifies) the root of the modification tree with the node in the other tree that is being modified (see Figure 8.1). Thus the sister ad-

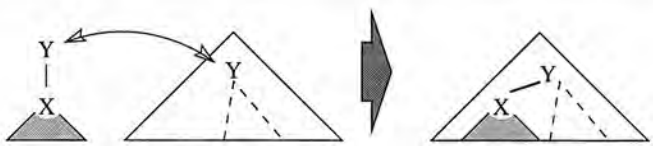


Figure 8.1: The new sister-adjunction

junction constraints (SACs) can be stated as features and modification trees can be ruled out by failure of unification. Our generation work provides additional motivation for introducing the new root of the modification trees, namely that the applicability semantics for modification mapping rules have a head concept which is not covered (consumed) by the construction and which is not linked to any syntactic node. This concept was used in finding where to attach (sister-adjoin) the modification tree of the mapping rule. Operations like the new sister-adjunction have been used in generation previously. Segment Grammar (SG) [Kempen 87a, Kempen & Hoenkamp 87] which assumes as basic structures (called segments) minimal trees consisting of a mother node (root) and a single daughter (called foot) uses a combination operation which merges two nodes of two segments (or derived structures). When the foot of a segment s_1 is merged with the root of a segment s_2 we have the standard substitution. When the roots of two segments are merged we have what amounts to the new sister adjunction

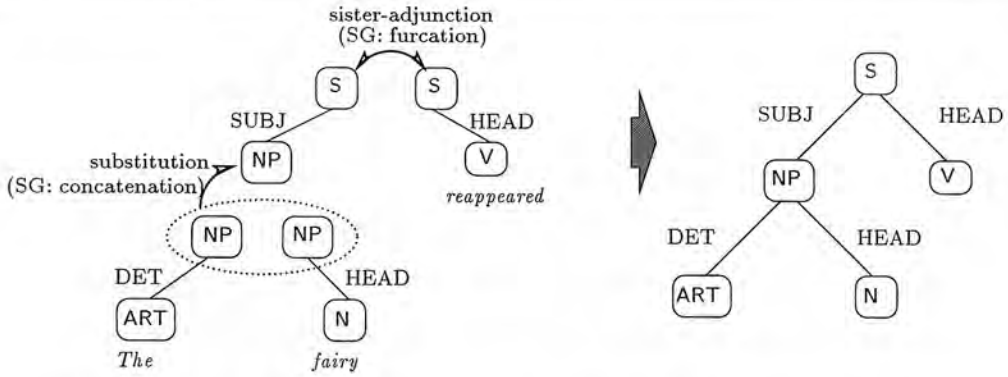


Figure 8.2: Previous use of sister-adjunction in segment grammar

(see Figure 8.2). Segment Grammar was used as the basis of the IPF generation system [De Smedt 90, page 169].

Adding given-new, focus-topic information: This could be handled either by adding additional fields to the mapping rules or using the existing mechanisms for encoding such information in the applicability 'semantics' (which will no longer have only semantic character). The former approach is more elegant.

Extending the grammar coverage: As it stands our generation system can be easily extended to incorporate more mapping rules covering more syntactic constructions.

Building supporting tools: Our goal in building the generation system was not to spend much effort on interface issues (as they are not central to the research contributions of this thesis). The system does have facilities for graphical display of the syntactic structure and more supporting utilities are being built.[2] Building a specialised mapping rules editor that can be used by the linguist will greatly enhance the usability of the system. There are new tools becoming available both for displaying conceptual graphs and syntactic trees and we intend to concentrate

[2] PROTECTOR-95 has a graphical display based on SICStus Prolog's 2.9 graphics manager which is no longer supported in SICStus v.3 and higher. The current system uses LIFE's own graphical libraries. We had made small experiments with Tcl/Tk though future development will be based on the emerging Java graphical toolkit.

on integrating existing resources rather than developing new ones from scratch.

8.3.2 Longer-term improvements

Follow-up sentences: None of the existing generation systems tackle the problem of generating follow-up sentences as a consequence of the assumption that the input is of a clause size and can be expressed as a single (though maybe complex) sentence. Sometimes generators can start off following a branch of the search space leading to a sub-optimal solution which does not allow the incorporation of all the semantic material in a sentence (this is a very likely scenario in left-to-right generation). Instead of backtracking which has the risks of not being able to reuse constituents that have been generated a generator could finish the current sentence, and replan the next sentence by carrying over unexpressed semantic information. The notion of constructing follow-up sentences was suggested in [Nogier & Zock 92]. One approach to constructing the conceptual input for the follow-up sentence would be to take the current input semantics, and remove all concepts that have been expressed but the ones that preserve connectivity between unexpressed ones. In order to avoid repetitions in the latter sentence(s) the generator will need to deal with anaphoric references and elliptical constructions which will account for the concepts expressed in the previous sentence. Generating follow-up sentences is another way to increase the paraphrasing power of the generator. Levelt's discussion for planning the conceptual input in the context of a dialogue is also relevant here [Levelt 89].

Move-α generation: The idea is that the generator only generates a basic structure (i.e., what Transformational Grammarians would call *deep structure*) and then (purely syntactic) lexical rules are used to derive other paraphrases from this. This idea is based on the observation that tree families in a DTG grammar have very similar trees in the sense the syntactic function of individual constituents is the same across many of the trees in a family and only the structural organisation is different. For example, NP constituents in sentential trees have the same case (see Figure 8.3). In the presently biggest DTG grammar there are about 50 basic trees (families) which have about 1800 trees together. Computationally it is

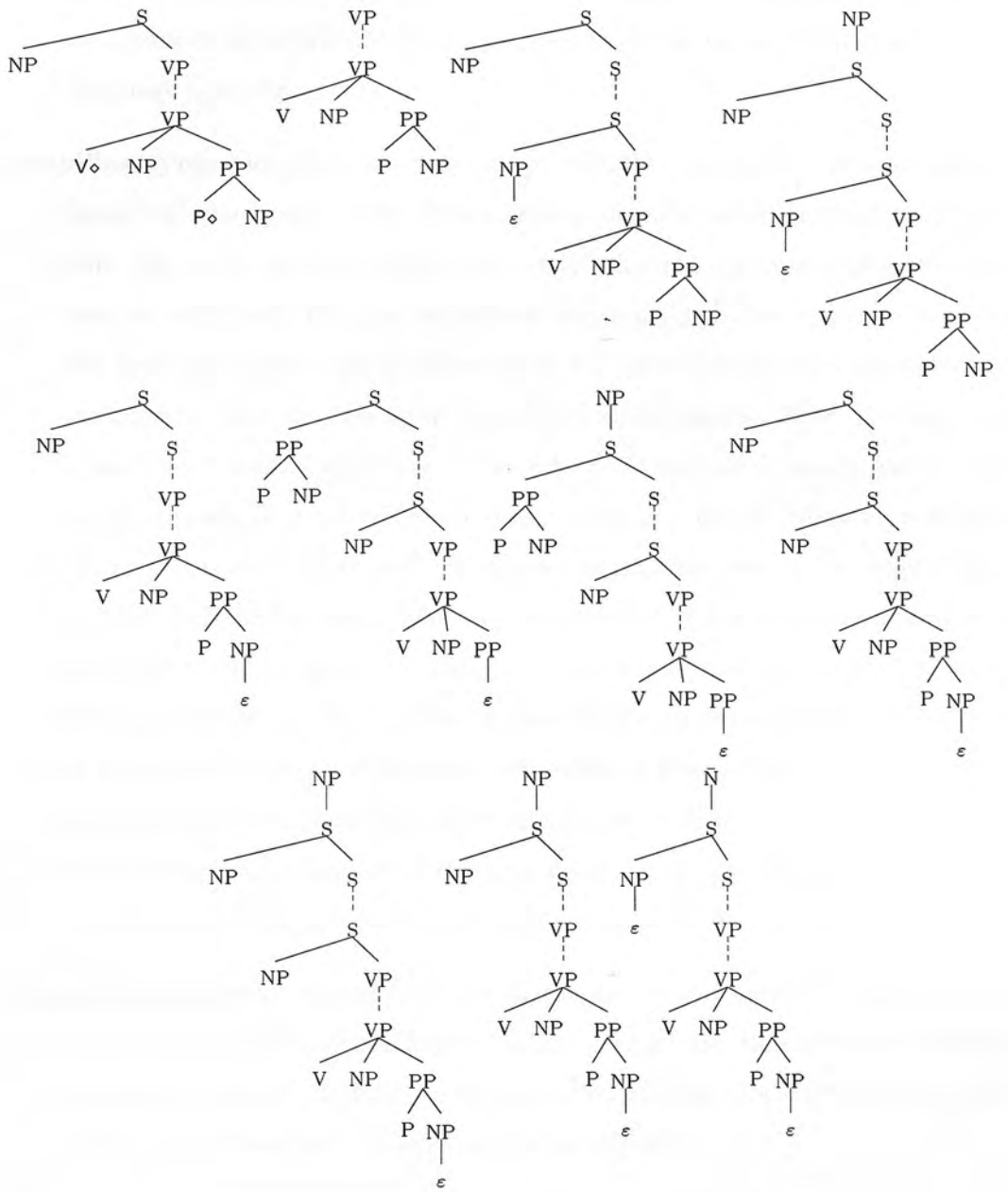


Figure 8.3: Trees from the `V_NP_PP_to` family

much cheaper to derive a paraphrase syntactically as opposed to going through a heavy full-blown generation. Yet, mechanisms like making the derivational rules that derive structures from the base tree sensitive to constraints like topic/focus are needed in order for the generator to distinguish the applicability of different syntactic structures. Such an approach might be useful in Computer-Aided Language Learning (CALL).

Compiling syntactic patterns: Compiling syntactic patterns is a pre-compilation technique[3] which can benefit from a good preference-based approach to generation. The main idea is to combine mapping rules into bigger mapping rules and operate with these. This can be justified from a psycholinguistic perspective and also from engineering considerations [Zock 97]. Good candidates for combination are mapping rules that are always or 'often'[4] used together. Thus, this approach looks at optimisation above the sentence level. Compiling syntactic patterns depends radically on good preference-based techniques for the following reason: If we do preserve completeness[5] we will not change the size of the search space but the shape of the search space will be changed. It will be flatter and at each stage there will be more alternatives. Thus it is crucial for a generator using this approach to be able to identify good partial structures quickly. Note that in choosing a syntactic framework with enlarged domain of locality we already have some of that syntactic pattern compilation built in. The overall approach can be seen as an extension of the counterpart of the technique for compiling a TAG grammar from an HPSG grammar [Kasper *et al.* 95].

Compiling large HPSG grammars to DTG format: There have been initial attempts to do a conversion from HPSG to TAG [Kasper *et al.* 95]. Researchers involved in this HPSG to LTAG compilation project see DTG as being more suitable than LTAG as the target formalism of their compilation algorithm.

---

[3] which could be combined with the hierarchy of mapping rules.

[4] It is clear what the meaning of rules used always together is. As for defining rules used frequently together we need to consider a domain, a task and sizable collection of generation goals.

[5] Similar approaches in parsing give up completeness, i.e., they undergenerate and gain good performance improvements sacrificing coverage of a small percentage of the corpus data.

Incremental generation: Even as it stands our generator offers benefits for incremental processing. The generator first finds a skeletal syntactic structure (phase 1) which is augmented later on in phase 2. In principle, if interrupted in phase 2 the generator would have a reasonable syntactic structure to return. Except for generators that have been built with incremental processing as their main objective this is not the case for the mainstream approaches to surface realisation.

Multilinguality: Both the use of a non-hierarchical semantic representation formalism and D-Tree Grammars with their uniform treatment of the operations on the syntactic and semantic side make our framework particularly well suited for multilingual generation.[6]

Exploring parallelism: There is a vast body of work within the area of Logic Programming which has studied parallel processing and a lot of these techniques are applicable to NLP and generation in particular. For instance, we could: (1) explore all possible applicable mapping rules simultaneously (OR-parallelism); (2) having chosen a mapping rule which has a lexicalised elementary syntactic structure generate in parallel all constituents (in case of interdependencies between siblings we need to synchronise the processes).

User modelling: Incorporating a model of the user would be a valuable addition to PROTECTOR. The lexical resources and the syntactic constructions need to annotated with features specifying for what kind of users they are applicable. Checking these can easily be incorporated in the generation process.

Large scale generation: In comparison to the biggest NLG systems[7] PROTECTOR remains a relatively small one. There are a number of issues that will arise when attempting to scale PROTECTOR:

- speed of matching
- indexing of rules

[6] Owen Rambow has investigated treatments of German phenomena in DTG and presented this work at the TAG+4 workshop in Philadelphia, Aug'98. For the predecessor of DTG, TAG, there exist large grammars for French [Abeillé 91] and German [Harbusch *et al.* 91].

[7] Modern, large NLG systems have ontologies of 50,000 concepts and lexica of 100,000 words and phrases.

- grammar development
- maintaining lexical resources

To sum up we developed a generation framework which addresses issues that have not been studied previously. Many of the ideas in this thesis are gradually becoming the focus of more and more work in the generation community. More people are considering generation from non-hierarchical structures (though sometimes not having concrete motivation to do so); more people are considering declarative architectures for generation, even techniques for converting parsing grammars into formats suitable for generation are being studied; deviations from the input semantics are also considered (e.g., negotiation architecture for MT); more people are looking at memoization techniques (though on all occasions researchers' motivation has been to efficiently produce all paraphrases and the issue that such mechanisms are important even for single sentence generation has passed unnoticed); preference-based generation is also an area receiving more attention.

A major effort behind this research was also to integrate the semantic and syntactic processing that allows consideration of alternative processing strategies for generation. The uniform treatment of the syntactic operations on the semantic side within our syntactic framework allows for the vast body of parsing techniques to be re-interpreted from a generation perspective.

Bibliography

[Abeillé 91] Anne Abeillé. *Une grammaire lexicalisée d'arbre adjoint pour le Français.* Unpublished PhD thesis, University Paris 7, France, 1991.

[Adorni & Zock 96] Giovanni Adorni and Michael Zock, editors. *Trends in Natural Language Generation: An Artificial Intelligence Perspective.* Springer Verlag, Heidelberg, 1996.

[Aho *et al.* 83] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *Data Structures and Algorithms.* Addison-Wesley, Reading, Mass., 1983.

[Aho *et al.* 86] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools.* Addison-Wesley, Reading, Mass., 1986.

[Aït-Kaci & Podelski 93] Hassan Aït-Kaci and Andreas Podelski. Towards a meaning of LIFE. *Journal of Logic Programming*, 16(3&4):195–234, July-August 1993.

[Aït-Kaci *et al.* 89] Hassan Aït-Kaci, R. Boyer, P. Lincoln, and R. Nasr. Efficient implementation of lattice operations. *ACM Transactions on Programming Languages and Systems*, 11(1):115–146, 1989.

[Al-Jabri 97] Saad Al-Jabri. *Generating Arabic Words from Semantic Descriptions.* Unpublished PhD thesis, University of Edinburgh, 1997.

[Alshawi 92] H. Alshawi, editor. *The Core Language Engine.* MIT Press, Cambridge, Mass., 1992.

[Antonacci & Smith 92] F. Antonacci and J. Smith. Analysis and Generation of Italian Sentences. In T. Nagle, J. Nagle, L. Gerholz, and P. Eklund, editors, *Conceptual Structures: Current research and Practice*, pages 437–460. Ellis Horwood, 1992.

[Appelt 85] D. E. Appelt. *Planning English Sentences.* Studies in Natural Language Processing. Cambridge University Press, Cambridge, 1985.

[Baader & Nipkow 98] Franz Baader and Tobias Nipkow. *Term Rewriting and All That.* Cambridge university Press, Cambridge, 1998.

[Backofen 94] Rolf Backofen. *Expressivity and Decidability of First-order Languages over Feature Trees.* Unpublished PhD thesis, Universität des Saarlandes, Saarbrücken, Germany, December 1994.

[Barnett *et al.* 94] James Barnett, Inderjeet Mani, and Elaine Rich. Reversible machine translation: What to do when the languages don't match up. In Tomek Sztralkowski, editor, *Reversible Grammar in Natural Language Processing.* Kluwer, Dordrecht, 1994.

[Barwise & Perry 83] Jon Barwise and John Perry. *Situations and Attitutes.* MIT Press, Cambridge, Mass., 1983.

[Barwise *et al.* 91] Jon Barwise, Jean Mark Gawron, Gordon Plotkin, and Syun Tutiya. *Situations Theory and Its Applications.* CSLI, Stanford, Calif., 1991.

[Bateman 96] John A. Bateman. KPML Development Environment: Multilingual linguistic resource development and sentence generation. Technical report, German National Center for Information Technology (GMD), Institute for Intergrated publication and information systems (IPSI), Darmstadt, Germany, 1996. URL: http://www.stir.ac.uk/english/communication/Computational-tools/kpml.html.

[Bateman *et al.* 89] John A. Bateman, J. R. Kasper, Joanna Moore, and R. Whitney. A general organization for knowledge for natural language processing: The PENMAN upper model. Technical report, USC/Information Science Institute, 1989.

[Bateman *et al.* 91] John A. Bateman, Elizabeth A. Maier, Elke Teich, and Leo Wanner. Towards an architecture for situated text generation. In *Proceedings of the International Conference on Current Issues in Computational Linguistics*, volume 2, pages 966–971, Penang, Malaysia, 1991.

[Becker *et al.* 98] Tilman Becker, Wolfgang Finkler, Anne Kilger, and Peter Poller. An efficient kernel for multilingual generation in speech-to-speech dialogue translation. In *Proceedings of COLING-ACL'98)*, Montreal, Canada, 1998.

[Bellos 92] Illona Bellos. Towards a multilingual IDAS. Unpublished M.Sc. thesis, University of Edinburgh, Scotland, 1992.

[Black & Taylor 97] Alan Black and Paul Taylor. Festival speech synthesis system: system documentation (1.1.1). Technical report HCRC/TR-83, Human Communication Research Centre, University of Edinburgh, UK, 1997.

[Bontcheva 96] Kalina Bontcheva. Generation of multilingual explanations from conceptual graphs. In R.Mitkov and N.Nicolov, editors, *Recent Advances in Natural Language Processing*. John Benjamins, Amsterdam, 1996. (to appear).

[Book & Otto 93] Ronald V. Book and Friedrich Otto. *String-Rewriting Systems*. Texts and Monographs in Computer Science. Springer-Verlag, Berlin, 1993.

[Boyer & Lapalme 85] Michel Boyer and Guy Lapalme. Generating paraphrases from meaning-text semantic networks. *Computational Intelligence*, 1(1):103–117, 1985.

[Brachman & Schmolze 85] Ronald Brachman and James Schmolze. An overview of the KL-ONE knowledge representation system. *Cogntive Science*, 4(9):171–216, 1985.

[Bresnan & Kaplan 82] Joan Bresnan and Ronald M. Kaplan. Lexical-Functional Grammar: A formal system for grammatical representation. In Joan Bresnan, editor, *The mental representation of grammatical relations*, Cognitive Theory and Mental Representation, chapter 4, pages 173–281. MIT Press, Cambridge, Mass., 1982.

[Bresnan 82] Joan Bresnan. *The mental representation of grammatical relations*. Cognitive Theory and Mental Representation. MIT Press, Cambridge, Mass., 1982.

[Burchfield 96] Robert W. Burchfield, editor. *The New Fowler's Modern English Usage*. Oxford University Press, 1996. third edition.

[Busemann 89] Stephan Busemann. Generation strategies for GPSG. In *Extended Abstracts presented at the Second European Natural Language Generation Workshop*, pages 105–110, Edinburgh, April 6–8 1989.

[Carpenter 92] Bob Carpenter. *The Logic of Typed Feature Structures*. Cambridge University Press, Cambridge, England, 1992.

[Carroll *et al.* 98] John Carroll, Nicolas Nicolov, Olga Shaumyan, Martine Smets, and David Weir. LexSys Project. In *Proceedings of the 4th International Workshop on Tree-adjoining Grammars and Related Frameworks*

(TAG+'98), pages 29–33, Philadelphia, USA, 1–3 August 1998.

[Consortium 96] GIST Consortium. GIST (Generating InStructional Text. Technical report, University of Brighton, 1996. Final report.

[Copestake *et al.* 97] Ann Copestake, Dan Flickinger, and Ivan A. Sag. Minimal recursion semantics: an introduction. (draft), 1997.

[Cormen *et al.* 90] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms.* The MIT Electrical Engineering and Computer Science Series. The MIT Press, Cambridge, Massachussetts, 1990.

[Covington 94] Michael A. Covington. *Natural Language Processing for Prolog Programmers.* Prentice-Hall, Englewood Cliffs, New Jersey 07632, 1994.

[Cristea 93] Dan Cristea. Romanian morphology generation through classification. Research report, DAI, University of Edinburgh, UK, 1993.

[Cyc96] Cycorp, Cycorp, Inc., 3500 West Balcones Center Drive, Austin, TX 78759. *Upper Cyc Ontology*, beta-version release edition, 3 August 1996.

[Dale 93] Robert Dale. Natural language generation. ESSLLI Summer School Notes, Madrid, 16–27 August 1993.

[Danlos & Meunier 96] Laurence Danlos and F. Meunier. G-tag, un formalisme pour la génération de textes: présentation et applications industrielles. In *Actes du colloque Informatique et Langue Naturelle.* Nantes, France, 1996.

[Davey 72] A. C. Davey. *A Computational Model of Discourse Production.* Unpublished PhD thesis, University of Edinburgh, 1972.

[Davey 78] A. C. Davey. *Discourse Production: A Computer Model of Some Aspects of a Speaker.* Edinburgh University Press, Edinburgh, 1978.

[De Smedt 90] Koenraad J.M.J. De Smedt. An incremental parallel formulator. In *Current Research in Natural Language Generation*, Cognitive Science Series, pages 167–193, London, 1990. Academic Press.

[Delugach 92] Harry S. Delugach. An exploration into semantic distance. In Heather D. Pfeiffer and Timothy E. Nagle, editors, *Conceptual Structures: Theory and Implementation*, pages 119–124. Springer-Verlag (LNAI 754), Las Cruces, NM, USA, 8–10 July 1992. Proceedings of the 7th Annual Workshop on Conceptual Structures.

[Dick 93] Collin Dick. Classification-based language generation in Turkish. Unpublished M.Sc. thesis, University of Edinburgh, Scotland, 1993.

[Doran *et al.* 94] Christine Doran, Dania Egedi, Beth Ann Hockey, Bangalore Srinivas, and Martin Zaidel. XTAG — A Wide Coverage Grammar for English. In *Proceedings of the 15th Int. Conference on Computational Linguistics (COLING-94)*, pages 922–928, Kyoto, Japan, 5-9 August 1994.

[Dorna *et al.* 98] Michael Dorna, Anette Frank, Josef van Genabith, and Martin Emele. Syntactic and semantic transfer with f-structures. In *Proceedings of the 17th International Conference on Computational Linguistics and the 36th Annual Meeting of the Association for Computational Linguistics (COLING-ACL'98)*, Montreal, Canada, 1998.

[Dorr 93] Bonnie J. Dorr. Interlingual machine translation: a parameterized approach. *Artificial Intelligence*, 63(1–2):429–492, October 1993.

[Dorr 94] Bonnie J. Dorr. Machine Translation Divergences: A Formal Description and Proposed Solution. *Computational Linguistics*, 20(4):597–633, 1994.

[Dörre & Dorna 93] Jochen Dörre and Michael Dorna. CUF – A Formalism for Linguistic Knowledge Representation. ESPRIT Basic Research Action BR3175 DYANA 2 Deliverable R1.2.A, Institut für Maschinelle Sprachverarbeitung, Universität Stuttgart, Azenbergstr. 12, 70174 Stuttgart, Germany, August 1993.

[Dowty 91] David Dowty. Thematic proto-roles and argument selection. *Language, Journal of the Linguistic Society of America*, 67:547–619, 1991.

[Earley 70] Jay Earley. An efficient context-free parsing algorithm. *CACM*, 6(8):451–455, 1970.

[Elhadad 91] Michael Elhadad. FUF: the universal unifier, user manual, ver 5.0. Technical report, Columbia University, Dept. of CS, New York 10027, October 1991.

[Elhadad 93] Michael Elhadad. *Using Argumentation to Control Lexical Choice: A Functional Unification Implementation.* Unpublished PhD thesis, Graduate School of Arts and Sciences, Columbia University, 1993.

[Ellis 94] R. Ellis. *The Study of Second Language Acquisition.* Oxford University Press, 1994.

[Ellis 95] Gerard Ellis. Compiling conceptual graphs. *IEEE Transactions on Knowledge and Data Engineering*, 7(1), February 1995.

[Ellis 97] R. Ellis. *Second Language Acquisition Research and Language Teaching.* Oxford University Press, 1997.

[Fall 95] Andrew Fall. Spanning Tree Representations of Graphs and Orders in Conceptual Structures. In Gerard Ellis, Robert Levinson, William Rich, and John Sowa, editors, *LNAI 954, Conceptual Structures: Applications, Implementation and Theory*, pages 232–246. Springer-Verlag, Berlin, 14–18 August 1995. Proceedings of the 3rd Int. Conf. on Conceptual Structures, (ICCS'95), Santa Cruz, CA, USA.

[Fawcett & Tucker 90] Robin P. Fawcett and Gordon H. Tucker. Demonstration of GENESYS: a very large, semantically based. In *Proceedings of the 13th International Conference on Computational Linguistics (COLING-90)*, volume 1, pages 47–49, Helsinki, Finland, 1990. Academic Press.

[Firth 57] J. Firth. The technique of semantics. In *Papers in linguistics 1934–1951*, pages 7–33. Oxford University Press, London, 1957. original published in 1935.

[Foo *et al.* 89] Norman Foo, B. Garner, A. Rao, and E. Tsui. Semantic distance in conceptual graphs. In Janice A. Nagle and Timothy E. Nagle, editors, *Fourth Annual Workshop on Conceptual Structures*, 1989.

[Garey & Johnson 79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* A series of books in the Mathematical Sciences. W. H. Freeman and Company, New York, 1979.

[Garner *et al.* 87] Brian Garner, Dickson Lukose, and E. Tsui. Parsing natural language through pattern correlation and modification. In *Proceedings of the 7th International Workshop on Expert Systems and their Applications*, pages 1285–1299, Avignon, France, 13–15 May 1987.

[Gazdar & Mellish 89] Gerald Gazdar and Chris Mellish. *Natural Language Processing in Prolog: An Introduction to Computational Linguistics.* Addison-Wesley, Wokingham, England, 1989.

[Gerdemann & Hinrichs 95] Dale Gerdemann and Erhard Hinrichs. Some open problems in head-driven generation. In Jerry Morgan, Georgia Green, and Jennifer Cole, editors, *Linguistics & Computation*, pages 65–197. CSLI Publications, 1995.

[Gromova *et al.* 96] Nevena Gromova, Chris Mellish, and Nicolas Nicolov. Sentence generation in russian using classification. draft, 1996.

[Grosz & Sidner 86] Barbara Grosz and Candance Sidner. Attention, intention, and the structure of discourse. *Computational Linguistics*, 12:175–204, 1986.

[Halliday 94] M. A. K. Halliday. *An Introduction to Functional Grammar.* Edward Arnold, 2nd edition, 1994.

[Harbusch *et al.* 91] Karin Harbusch, Wolfgang Funkler, and Anne Schauder. Incremental syntax generation with tree adjoining grammars. Technical report, DFKI, Saarbrücken, Germany, 1991. RR-91-25.

[Harbusch *et al.* 94] K. Harbusch, G. Kikui, and A. Kilger. Default handling in incremental generation. In *Proceedings of the 15th International Conference on Computational Linguistics (COLING-94)*, pages 356–362, Kyoto, Japan, 1994.

[Harrison 78] Michael A. Harrison. *Introduction to Formal Language Theory.* Addison-Wesley Series in Computer Science. Addison-Wesley, Reading, Mass., 1978.

[Hartshorne *et al.* 58] C. Hartshorne, P. Weiss, and A. Burks, editors. *Collected Papers of C. S. Pierce, 8 vols.* Harvard University Press, Cambridge, Mass., 1958. 1931–1958.

[Haruno *et al.* 93] Masahiko Haruno, Yasuharu Den, Yuji Matsumoto, and Makoto Nagao. Bidirectional chart generation of natural language texts. In *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI'93)*, pages 350–356, Menlo Park, CA, 1993. American Association for Artificial Intelligence, The MIT Press.

[Helman & Veroff 86] Paul Helman and Robert Veroff. *Intermediate Problem Solving and Data Structures: Walls and Mirrors.* Benjamin/Cummings Series in Structured Programming.

The Benjamin/Cummings Publishing Company, Wokingham, U.K., 1986.

[Hogger 91] Christopher John Hogger. *Essentials of Logic Programming.* Graduate Texts in Computer Science. Clarendon Press, Oxford, 1991.

[Hopcroft & Ullman 79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley Series in Computer Science. Addison-Wesley, Reading, Mass., 1979.

[Iordanskaja & *et al.* 91] Lidija Iordanskaja *et al.* Lexical Selection and Paraphrase in a Meaning-Text Generation Model. In Cécile Paris, William Swartout, and William Mann, editors, *Natural Language Generation in Artificial Intelligence and Computational Linguistics*, pages 293–312. Kluwer Academic, 1991.

[Jackendoff 95] Ray Jackendoff. The architecture of the language faculty. 1995.

[Janes 88] Michael Janes. *Harrap's Dictionnaire Anglais-Français.* Harrap Books Ltd, Edinburgh, 1988. p317.

[Johnson 85] Mark Johnson. Parsing with discontinuous constituents. In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*, 1985.

[Joshi & Schabes 92] Aravind K. Joshi and Yves Schabes. Tree-adjoining grammars and lexicalized grammars. In Maurice Nivat and Andreas Podelski, editors, *Definability and Recognizability of Sets of Trees.* Elsevier, 1992.

[Joshi 85] Aravind K. Joshi. How much context-sensitivity is necessary for characterizing structural descriptions—tree adjoining grammars. In D.Dowty, L.Karttunen, and A.Zwicky, editors, *Natural Language Processing—Theoretical, Computational and Psychological Perspective*, chapter 16, pages xx–xx. Cambridge University Press, New York, 1985.

[Joshi 87] Aravind Joshi. The Relevance of Tree Adjoining Grammar to Generation. In Gerard Kempen, editor, *Natural Language Generation*, pages 233–252. Kluwer Academic, Dordrecht, The Netherlands, 1987.

[Joshi *et al.* 75] Aravind K. Joshi, L.S. Levi, and M. Takahashi. Tree adjoining grammars. *Journal of Comput. Syst. Sci.*, 10(1):xx–xx, 1975.

[Kameyama *et al.* 91] Megumi Kameyama, Ryo Ochitani, and Satanley Peters. Resolving translation mismatches with information flow. In *Annual Meeting of the Association of Computational Linguistics*, pages 193–200, Berkeley, California, 1991. Association of Computational Linguistics.

[Kamp & Reyle 93] Hans Kamp and Uwe Reyle. *From Discourse to Logic.* Kluwer Academic Publishers, London, 1993.

[Kasper *et al.* 95] Robert Kasper, Bernd Kiefer, Klaus Netter, and K. Vijay-Shanker. Compilation of HPSG to TAG. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, MIT, Cambridge, Mass., 26–30 June 1995.

[Kay 79] Martin Kay. Functional grammar. In C. Chiarello et al., editor, *Proceedings of the Fifth Annual Meeting of the Berkeley Linguistics Society*, Berkeley, California, 1979.

[Kay 80] Martin Kay. Algorithm schemata and data structure in syntactic processing. Technical Report CLS-80-12, Xerox PARC, 1980.

[Kay 83] Martin Kay. Unification grammar. Technical report, Xerox Palo Alto Research Center (PARC), Palo Alto, California, 1983.

[Kay 84] Martin Kay. Functional unification grammar: a formalism for machine translation. In *Proceedings of the 10th International Conference on Computational Linguistics, COLING'84*, Stanford, California, 1984.

[Kay 85] Martin Kay. Parsing in functional unification grammar. In D. R. Dowty, Lauri Karttunen, and A. Zwicky, editors, *Natural Language Parsing*, pages 206–250. Cambridge University Press, Cambridge, 1985.

[Kay 96] Martin Kay. Chart generation. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linquistics (ACL'96)*, pages 200–204, Santa Cruz, California, 1996.

[Kempen & Hoenkamp 87] Gerard Kempen and E. Hoenkamp. An incremental procedural grammar for sentence formulation. *Cognitive Science*, 11:201–258, 1987.

[Kempen 87a] Gerard Kempen. A framework for incremental syntactic tree formation. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence,*

Milan, Italy, pages 655–660, Los Altos, Calif., 1987. Morgan Kaufmann.

[Kempen 87b] Gerard Kempen. *Natural Lanaguage Generation: Recent Advances in Artifiacial Intelligence, Psychology and Linguistics.* Kluwer Academic, The Netherlands, 1987.

[Klein 96] Marion Klein. Towards a multilingual IDAS. Unpublished M.Sc. thesis, University of Edinburgh, Scotland, 1996.

[Knight & Hatzivassiloglou 95] Kevin Knight and Vasileios Hatzivassiloglou. Two-level, many-paths generation. In *Proceedings of the 33rd Annual Meeting of the ACL (ACL'95)*, pages 252–260, Cambridge, Massachusetts, 1995.

[Knott 96] Alistair Knott. *A Data-Driven Methodology for Motivating a Set of Coherence Relations.* Unpublished PhD thesis, University of Edinbugh, Edinbugh, U.K., 1996.

[Kohl & Momma 92] Dieter Kohl and Stefan Momma. In Gabriel Bes, editor, *The Construction of a Natural Language and Graphic Interface—Results and perspectives from the ACCORD project*, part generation in accord 5. Springer, 1992.

[Kohl 91] Dieter Kohl. Generierung aus unter- und überspezifizierten merkmalsstrukturen in LFG. Arbeitspapiere für die Computerlinguistik Bericht 9, Institut für Maschinelle Sprachverarbeitung, 1991.

[Kohl 92] Dieter Kohl. Generation from under- and overspecified structures. In Christian Boitet, editor, *Proceedings of the 15th International Conference on Computational Linguistics*, volume 2, pages 686–692, Nantes, France, August 1992.

[Kolodner 93] Janet L. Kolodner. *Case-Based Reasoning.* Morgan Kaufmann, 1993.

[König 94] Esther König. Syntactic head-driven generation. In *Proceedings of the 15th Int. Conference on Computational Linguistics (COLING'94)*, 475-481, Kyoto, 1994.

[Kroch & Joshi 85] A. Kroch and A. K. Joshi. Linguistic relevance of tree adjoining grammars. Technical report, Dept. of CS, University of Pennsylvania, Philadelphia, 1985. MS-CIS-85-18.

[Kukich 88] Karen Kukich. Fluency in natural language reports. In David D. McDonald and Leonard Bolc, editors, *Natural Language Generation Systems.* Springer, Berlin, 1988.

[Levelt 89] Willem J. M. Levelt. *Speaking: From Intention to Articulation.* A Bradford Book, The MIT Press, London, England, 1989.

[Linden *et al.* 92] K. Vander Linden, S. Cumming, and J. Martin. Using system networks to build rhetorical structures. In Robert Dale, Ed Hovy, and Dietmar Rösner, editors, *Aspects of Automated Natural Language Generation*, pages 183–198. Springer-Verlag, 1992.

[MacGregor 90] R. MacGregor. LQOM user's manual. La Jolla, California, 1990.

[Malinowski 23] B. Malinowski. Supplement i: The meaning of meaning. Supplement to C.K. Ogden and I.A. Richards, Hartcourt Brace, 1923.

[Manandhar 94] Suresh Manandhar. An attributive logic of set descriptions and set operations. In *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics (ACL'94), Las Cruces, New Mexico, USA*, pages 255–262. HCRC Language Technology Group, Univ of Edinburgh, UK, 27-30 June 1994.

[Manber 89] Udi Manber. *Introduction to Algorithms: A Creative Approach.* Addison-Wesley, 1989.

[Mann & Matthiessen 85] William Mann and Christian M.I.M. Matthiessen. A demonstration of the Nigel text generation computer program. In James D. Benson and William S. Greaves, editors, *Systemic Perspectives on Discourse.* Ablex Norwood, N.J., 1985.

[Mann 83] William C. Mann. An overview of the PENMAN text generation system. Technical report, USC/Information Sciences Institute, Marina del Rey, Calif., 1983. Technical Report ISI/RR-83-114.

[Masahiko & Matsumoto 93] Yasuharu Den Masahiko and Yuji Matsumoto. Bidirectional chart generation algorithm. In *Proceeedings of the 4th European Workshop on Natural Language Generation*, Pisa, Italy, 1993.

[Matthiessen & Bateman 91] Christian M.I.M. Matthiessen and John A. Bateman. *Text Generation and Systemic-Fuctional Linguistics: Experiences from English and Japanese.* Pinter, London, 1991.

[Matthiessen 83] Christian M.I.M. Matthiessen. Systemmic grammar in computation: The NIGEL case. In *Proceedings of the 1st Conference of the European ACL (EACL'83)*, pages xx–yy, Pisa, Italy, 1-2 September 1983.

[McCoy *et al.* 92] Kathleen F. McCoy, K. Vijay-Shanker, and Gijoo Yang. A functional approach to generation with tag. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics(ACL'92)*, pages 48–55, University of Delaware, U.S., 1992. Association of Computational Linguistics.

[McDonald & Conklin 82] David D. McDonald and E. Jeffery Conklin. Salience as a simplifying metaphor for natural language generation. In *Proceedings of the National Conference on Artificial Intelligence (AAAI'82)*, pages 75–78. American Association for Artificial Intelligence, August 18–20 1982.

[McDonald & Pustejovsky 85] David McDonald and James Pustejovsky. TAGs as a grammatical formalism for generation. In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*, pages 94–103, 1985.

[McDonald 81] David McDonald. MUMBLE , a flexible tool for natural language production. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pages 1062–1062, Vancouver, B.C., 1981. University of British Columbia. volume II.

[McGregor 82] James J. McGregor. Backtrack search algorithms and the maximal common subgraph problem. *Software—Practice and Experience*, 1(1):23–34, 1982.

[McKeown & Swartout 88] Kathleen R. McKeown and William Swartout. Language generation and explanation. In Michael Zock and Gerard Sabah, editors, *Advances in Natural Generation: An Interdisciplinary Perspective*, volume 1. Pinter, London, 1988.

[McKeown 85] Kathleen R. McKeown. *Text Generation: Using discourse strategies and focus constraints to generate natural language text.* Studies in Natural Language Processing. Cambridge University Press, Cambridge, 1985.

[McKeown *et al.* 90] Kathleen R. McKeown, Michael Elhadad, Y. Fukumoto, J. Lim, C. Lombardi, Jaques Robin, and Frank Smadja. Language generation in COMET. In Robert Dale, Chris Mellish, and Michael Zock, editors, *Current Research in Natural Language Generation*, pages 103–140. Academic Press, London, 1990.

[Mel'cuk 88] Igor Aleksandrovich Mel'cuk. *Dependency Syntax: Theory and Practice.* University of New York Press, Albany, N.Y. State, 1988.

[Mellish & Reiter 93] Chris Mellish and Ehud Reiter. Using classification as a programming language. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI-93)*, volume 2, pages xx–yy, Chamberly, France, 1993.

[Mellish 88] Chris Mellish. Implementing systemic classification by unification. *Computational Linguistics*, 14(1):40–51, 1988.

[Mellish 91] Chris Mellish. Approaches to realization in natural language generation. In Ewan Klein and Frank Veltman, editors, *Natural Language and Speech*, ESPRIT Basic Research Series, pages 95–116. Springer-Verlag, London, November 26/27 1991. Simposium Proceedings, Brussels.

[Mellish 92] Chris Mellish. Term-encodable description spaces. In D. R. Brough, editor, *Logic Programming New Frontiers*, pages 189–207. Intellect, Oxford, 1992.

[Mellish 95] Chris Mellish. Techniques in natural language processing 1. Technical report, University of Edinburgh, 1995. Module Workbook.

[Mellish *et al.* 94] Chris Mellish, Roger Evans, David Allport, Lynne J. Cahill, Anthony F. Hartley, Robert Gaizauskas, and John Walker. The TIC message analyser. Technical report, University of Sussex, 1994.

[Meteer 90] Marie Wenzel Meteer. *The "Generation Gap": The Problem of Expressibility in Text Planning.* Unpublished PhD thesis, Computer and Information Science Department, University of Massachusetts, February 1990. COINS Technical Report 90-04.

[Meteer *et al.* 87] Marie Meteer, David D. McDonald, S. D. Anderson, D. Forster, L. S. Gay, A. K. Huettner, and P. Sibun. MUMBLE-86: Design and implementation. Technical report, University of Massachusetts at Amherst, Massachusetts, U.S.A., 1987. TR COINS 87-87.

[Minnen *et al.* 95] Guido Minnen, Dale Gerdemann, and Thilo Götz. Off-line Optimization for Earley-style hpsg Processing. In *Proceedings of the 7th Conference of the European Chapter of the Association for Computational Linguistics*, pages 173–179, Dublin, Ireland, 1995.

[Nicolov & Mellish 96] Nicolas Nicolov and Chris Mellish. Alternative semantic head driven generation for constraint-based grammars. draft, 1996.

[Nicolov *et al.* 95] Nicolas Nicolov, Chris Mellish, and Graeme Ritchie. Sentence Generation from Conceptual Graphs. In Gerard Ellis, Robert Levinson, William Rich, and John Sowa, editors, *LNAI 954, Conceptual Structures: Applications, Implementation and Theory*, pages 74–88. Springer, Berlin, 14–18 August 1995. Proceedings of the 3rd Int. Conf. on Conceptual Structures (ICCS'95), Santa Cruz, CA, USA.

[Nicolov *et al.* 96] Nicolas Nicolov, Chris Mellish, and Graeme Ritchie. Approximate Generation from Non-Hierarchical Representations. In *Proceedings of the 8th International Workshop on Natural Language Generation*, pages 31–40, Herstmonceux Castle, U.K., 13-15 June 1996.

[Nicolov *et al.* 97] Nicolas Nicolov, Chris Mellish, and Graeme Ritchie. Approximate Chart Generation from Non-Hierarchical Representations. In Ruslan Mitkov & Nicolas Nicolov, editor, *Recent Advances in Natural Language Processing: Selected papers from RANLP'95*, Current Issues in Linguistic Theory (CILT), 136, pages 273–294. John Benjamins, Amsterdam & Philadelphia, 1997.

[Nogier & Zock 92] Jean-François Nogier and Michael Zock. Lexical Choice as Pattern Matching. In Timothy E. Nagle, Janice A. Nagle, Laurie L. Gerholz, and Peter W. Eklund, editors, *Conceptual Structures: Current Research and Practice*, Ellis Horwood Series in Workshops, pages 413–436. Ellis Horwood, London, England, 1992.

[Nogier 90] Jean-François Nogier. *Un systèm de production de language fondé sur le modele des graphes conceptuels.* Unpublished PhD thesis, Paris VI, 1990.

[Nogier 91] Jean-François Nogier. *Génération automatique de langage et graphs conceptuels.* Hermès, Paris, 1991.

[Norvig 91] Peter Norvig. Techniques for Automatic Memoization with Application to Context-Free Parsing. *Computational Linguistics*, 17(1):91–98, 1991.

[Not & Pianta 95] Elena Not and E. Pianta. Issues of Multilinguality in the Automatic Generation of Administrative Instructional Texts. In M. Gori and G. Soda, editors, *Proceedings of the AI*IA '95 Congress*, pages xx–yy. Springer, 1995.

[O'Donnell 96] Michael O'Donnell. Input specification in the WAG sentence generation system. In *Proceedings of the 8th International Workshop on Natural Language Generation*, Herstmonceux Castle, UK, 13–15 June 1996.

[Oh *et al.* 92] Jonathan Oh, Steve Graham, Wen-Jung Hsin, Gi-Chul Yang, Young Bae Choi, Key-Sun Choi, and Sung-Hyon Myaeng. NLP: Natural Language Parsers and Generators. In G. Ellis and R. Levinson, editors, *Proc. of 1st Int. Workshop on PEIRCE: A Conceptual Graph Workbench*, pages 48–55, 1992.

[Ohmann 71] Richard Ohmann. In W. Morris, editor, *The Heritage Illustrated Dictionary of the English Language*, pages xxxi–xxxiv. American Heritage & Houghton Mifflin, New York, 1971.

[Paris & Scott 95] Cécile L. Paris and Donia Scott. DRAFTER: Support for the Production of Multilingual Instructions. In *Proceedings of the 2nd Language Engineering Convention*, pages 63–70, London, U.K., 1995.

[Paris *et al.* 91] Cécile L. Paris, William R. Swartout, and William R. Mann. *Natural Language Generation in Artificial Intelligence and Computational Linguistics: papers from the Forth International Text Generation Worshop.* Kluwer Academic, The Netherlands, 1991.

[Patten 88] Terry Patten. *Systemic Text Generation as Problem Solving.* Cambridge University Press, Cambridge, England, 1988.

[Patten *et al.* 92] Terry Patten, Michael L. Geis, and Barbara D. Becker. Toward a theory of compilation for natural language generation. *Computational Intelligence*, 8(1):77–101, 1992.

[Pianesi 93] Fabio Pianesi. Head-driven bottom-up generation and Government and Binding: a unified perspective. In Helmut Horacek and Michael Zock, editors, *New Concepts in Natural Language Generation*, chapter 3, pages 187–214. Pinter, London, 1993.

[Pierce 93] Charles Sanders Pierce. *Writings of Charles S. Pierce.* Indiana University Press, Bloomington, 1993. 1982–1993.

[Pollard & Sag 87] Carl Jesse Pollard and Ivan A. Sag. *Information Based Syntax and Semantics.* CSLI Lecture Notes 13, Chicago University Press, 1987.

[Pollard & Sag 94] Carl Jesse Pollard and Ivan A. Sag. *Head-Driven Phrase Structure Grammar.* Chicago University Press, 1994.

[Pollard 84] Carl Pollard. *Generalized Phrase Structure Grammars, Head Grammars and Natural Language.* Unpublished PhD thesis, Stanford University, 1984.

[Poole & Campbell 95] J. Poole and J. A. Campbell. A Novel Algorithm for Matching Conceptual and Related Graphs. In G. Ellis, editor, *Proceedings of the 3rd International Conference on Conceptual Structures.* Springer-Verlag, August 1995.

[Puder *et al.* 95] A. Puder, S.Markwitz, and F.Gudermann. Service Trading Using Conceptual Structures. In G. Ellis, editor, *Proceedings of the 3rd International Conference on Conceptual Structures.* Springer-Verlag, August 1995.

[Rambow *et al.* 95a] Owen Rambow, K. Vijay-Shanker, and David Weir. D-Tree Grammars. In *Proceedings of the 33rd Meeting of the Association for Computational Linguistics (ACL'95)*, pages 151–158, 1995.

[Rambow *et al.* 95b] Owen Rambow, K. Vijay-Shanker, and David Weir. Parsing D-Tree Grammars. In *Proceedings of the 4th ACL/SIGPARSE International Workshop on Parsing Technologies (IWPT'95)*, pages 252–259, 1995.

[Rambow *et al.* 96] Owen Rambow, K. Vijay-Shanker, and David Weir. Formal properties of d-tree grammars. unpublished manuscript, 1996.

[Reape 89] Mike Reape. A logical treatment of semi-free word order and bounded discontinuous constituency. In *Proceedings of the Forth Conference of the European Chapter of the Association for Computational Linguistics*, UMIST, Manchester, 1989.

[Reape 90] Mike Reape. Getting things in order. In *Proceedings of the Simposium on Discontinuous Constituency*, ITK, Tilburg, 1990.

[Reiter & Dale 92] Ehud Reiter and Robert Dale. A fast algorithm for the generation of referring expressions. In Christian Boitet, editor, *Proceedings of 14th International Conference on Computational Linguistics (COLING-92)*, volume 1, pages 232–238. MIT Press, 1992.

[Reiter & Dale 97] Ehud Reiter and Robert Dale. Building applied natural language generation systems. *Natural Language Engineering*, 3(1):57–87, 1997.

[Reiter & Mellish 92] Ehud Reiter and Chris Mellish. Using classification to generate text. In *Proceedings of the 30th Meeting of the Association for Computational Linguistics*, pages 265–272, Newark, University of Delaware, U.S.A., 28 June–2 July 1992.

[Reiter 91] Ehud Reiter. A new model of lexical choice for nouns. *Computational Intelligence*, 7(4):240–251, 1991. Special issue on Natural Language Generation.

[Reiter 95] Ehud Reiter. Nlg vs. templates. In *Proceedings of the Fifth European Workshop on Natural-Language Generation (ENLGW-1995)*, Leiden, The Netherlands, 1995.

[Reiter *et al.* 92] Ehud Reiter, Chris Mellish, and John Levine. Automatic generation of on-line documentation in the IDAS project. In *Proceedings of the Third Conference on Applied Natural Language Processing (ANLP-1992)*, pages 64–71, Trento, Italy, 1992.

[Reyle 93] Uwe Reyle. Dealing with ambiguities by underspecification: Construction, representation and deduction. *Journal of Semantics*, 10:123–179, 1993.

[Ritchie 97] Graeme Ritchie. Completeness conditions for mixed strategy bidirectional parsing. Research Paper 861, Dept. of Artificial Intelligence, University of Edinburgh, 1997.

[Rösner & Stede 94] Dietmar Rösner and Manfred Stede. Generating multilingual documnts from a knowledge base: The TECHDOC project. In *Proceedings of the 15th International Conference on Computational Linguistics (COLING-94)*, volume 1, pages 339–346, Kyoto, Japan, 1994.

[Saint-Dizier 89] Patrick Saint-Dizier. A generation method based on principles of government and binding theory. In *Extended Abstracts presented at the Second European Natural Language Generation Workshop*, pages 111–117, Edinburgh, April 6–8 1989.

[Samuelsson 97] Christer Samuelsson. Example-Based Optimization of Surface-Generation Tables. In Ruslan Mitkov & Nicolas Nicolov, editor, *Recent Advances in Natural Language Processing*, Current Issues in Linguistic Theory (CILT) No.136, pages 295–315. John Benjamins, Amsterdam/Philadelphia, 1997.

[Schöter 93] Andreas Schöter. Compiling feature structures into terms: An empirical study in prolog. Research Re-

port EUCCS-RP-1993-1, University of Edinburgh, Edinburgh, Scotland, March 1993.

[Seuren 93] Pieter Seuren. Translation Relations in Semantic Syntax. In *Proceedings of the Conference on Computational Linguistics in the Netherlands (CLIN)*, pages xx–xx, 1993.

[Shapiro & Rapaport 90] Stuart C. Shapiro and William J. Rapaport. The SNePS family. Technical report 90-21, Department of Computer Science, State University of New York at Buffalo, Buffalo, NY., 1990.

[Shapiro 82] Stuart Shapiro. Generalized augmented transition network grammars for generation from semantic networks. *American J. of Computational Linguistics*, 2(8):12–25, 1982.

[Shieber & Schabes 94] Stuart M. Shieber and Yves Schabes. An alternative conception of tree-adjoining derivation. *Computational Linguistics*, 20(1):91–124, 1994.

[Shieber 86] Stuart M. Shieber. *An Introduction to Unification-Based Approaches to Grammar.* Chocago University Press, Chicago, 1986.

[Shieber 88] Stuart M. Shieber. A uniform architecture for parsing and generation. In *COLING-88*, Budapest, 1988.

[Shieber 93] Stuart M. Shieber. The problem of logical-form equivalence. *Computational Linguistics*, 19(1):179–190, 1993.

[Shieber *et al.* 90] Stuart M. Shieber, Gertjan van Noord, Robert C. Moore, and Fernando C. N. Pereira. A semantic head-driven generation algorithm for unification-based formalisms. *Computational Linguistics*, 16(1):30–42, 1990.

[Simmons & Slocum 72] R. Simmons and J. Slocum. Generating English Discourse from Semantic Networks. *CACM*, 15(10):891–905, 1972.

[Skehan 98] P. Skehan. *A Cognitive Approach to Language Learning.* Oxford University Press, 1998.

[Smith *et al.* 94] Mark Smith, Roberto Garigliano, and Richard Morgan. Generation in the LOLITA system: an engineering approach. In *Proc. of 7th Int. Workshop on Natural Language Generation*, pages 241–244, 1994.

[Sowa 84] John Sowa. *Conceptual Structures: Information Processing in Mind and Machine.* Addison-Wesley, Reading, Massachusetts, 1984.

[Sowa 92] John F. Sowa. Conceptual graphs summary. In Timothy E. Nagle, Janice A. Nagle, Laurie L. Gerholz, and Peter W. Eklund, editors, *Conceptual Structures: Current Research and Practice*, Ellis Horwood Series in Workshops, pages 3–51. Ellis Horwood Limited, London, England, 1992.

[Sowa 93] John F. Sowa. Lexical structures and conceptual structures. In James Pustejovsky, editor, *Semantics and the Lexicon*, Studies in Linguistics and Philosophy, chapter III: Computational Models of Lexical Knowledge, pages 223–262. Kluwer Academic, London, 1993.

[Steel & DeRoeck 87] S. Steel and A. N. DeRoeck. Bidirectional chart parsing. In Chris Mellish and John Hallam, editors, *Advances in Artificial Intelligence.* Wiley, 1987.

[Stone & Doran 97] Matthew Stone and Christine Doran. Sentence Planning as Description Using Tree Adjoining Grammar. In *Proceedings of the 35th Annual Meeting of the Association of Computational Linguistics(ACL'97)*, pages 198–205, Madrid, Spain, 1997.

[Strzalkowski & Martinovic 92] Tomek Strzalkowski and Miroslav Martinovic. Comparing two grammar-based generation algorithms: A case study. In *Proceedings of the 30th Annual Meeting of the ACL*, pages 81–88, University of Delaware, U.S., 1992. Association of Computational Linguistics.

[Svenberg 94] Stefan Svenberg. Representing Conceptual and Linguistic Knowledge for Multilingual Generation in a Technical Domain. In *Proc. of 7th Int. Workshop on Natural Language Generation*, pages 245–248, 1994.

[Thompson & Ritchie 84] Henry Thompson and Graeme Ritchie. Implementing natural language parsers. In T. O'Shea and M. Eisenstadt, editors, *Artificial Intelligence: Tools, Techniques and Applications*, pages 245–300. Harper and Row, New York, 1984.

[Thompson 77] Henry Thompson. Strategy and tactics: A model for language production. In *Papers from the 13th Regional Meeting, Chicago Linguistic Society*, pages 651–668, Chicago, Illinois, 1977.

[vanNoord 90] Gertjan van Noord. An overview of head-driven bottom-up generation. In Robert Dale, Chris Mellish,

and Michael Zock, editors, *Current Research in Natural Language Generation*, Cognitive Science Series, pages 141–165. Academic Press, 1990.

[vanNoord 93] Gertjan van Noord. *Reversibility in Natural Languege Processing.* Unpublished PhD thesis, University og Utrecht, 1993.

[vanNoord 97] Gertjan van Noord. Efficient Head-Corner Parsing. *Computational Linguistics*, Vol(Num):xx–yy, 1997.

[vanNoord *et al.* 91] Gertjan van Noord, Joke Dorrepaal, Pim van der Eijk, Maria Florenza, Herbert Ruessnik, and Louis des Tombe. An overview of MiMo2. *Machine Translation*, (6):201–214, 1991.

[vanRijn 91] Afke van Rijn. *Natural Language Communication between Man and Machine.* Unpublished PhD thesis, Technical University Delft, 1991.

[Vijay-Shanker 92] K. Vijay-Shanker. Using Descriptions of Trees in a Tree Adjoining Grammar. *Computational Linguistics*, 18(4):481–517, 1992.

[Wahlster *et al.* 91] Wolfgang Wahlster, Elisabeth André Son Bandyopadhyay, Winfried Graf, and Thomas Rist. WIP: the coordinated generation of multimodal presentations from a common representation. RR 91-08, DFKI, 1991.

[Way 91] Eileen C. Way. Conceptual graph overview. *Journal of Experimental and Theoretical Artificial Intelligence (JETAI)*, 4(2):75–84, 1991. 6th Annual Workshop on Conceptual Graphs (July 1991).

[Wedekind 88] Jürgen Wedekind. Generation as structure driven derivation. In *Proceedings of the 12th International Conference on Computational Linguistics*, pages 732–737, Budapest, Hungary, August 1988.

[Wood 93] Mary McGee Wood. *Categorial Grammars.* Routledge, London, 1993.

[Wu & Palmer 94] Zhibiao Wu and Martha Palmer. Verb semantics and lexical selection. In *Proceedings of ACL'94*, 1994.

[Yang *et al.* 92] Gi-Chul Yang, Young Bae Choi, and Jonathan C. Oh. Cgma: A novel conceptual graph matching algorithm. In Heather D. Pfeiffer and Timothy E. Nagle, editors, *Conceptual Structures: Theory and Implementation*, pages 252–261. Springer-Verlag (LNAI 754), Las Cruces, NM, USA, 8–10 July 1992. Proceedings of the 7th Annual Workshop on Conceptual Structures.

[Zock 87] Michael Zock. Proposal for simulating on-line generation by giving priority either to lexical choices or syntactic structure. In *COGNITIVA*, 1987.

[Zock 90] Michael Zock. La génération interactive de langage: comment visualiser le passage de l'idée à la phrase. In J. Anis & J.L.Lebrave, editor, *Texte et ordinateurs: les mutation du lire-écrire.* Centre de Recherches Linguistique de Paris X, Nanterre, 1990.

[Zock 93] Michael Zock. Natural Language Generation. AIED'93 NLG Workshop Notes, Edinburgh, UK, August 1993.

[Zock 94] Michael Zock. Natural Language Generation. COLING-94 NLG Workshop Notes, Kyoto, Japan, August 1994.

[Zock 97] Michael Zock. Sentence Generation by Pattern Matching: The Problem of Syntactic Choice. In Ruslan Mitkov and Nicolas Nicolov, editors, *Recent Advances in Natural Language Processing 1995*, number 136 in Current Issues in Linguistic Theory, pages 317–352. John Benjamins, Amsterdam, 1997.

Appendix A

Generation Systems

In this appendix we list different generation systems that have been built over the years. Our list is largely based on [Adorni & Zock 96] with some additions and represents the most comprehensive collection of built generators up to date.

NAME	YEAR	LANGUAGE	GENERATOR	DESCRIPTION
Yngve	1961	English		random generation with a generative grammar
Klein	1965	English		story generator
Weizenbaum	1966	English	ELIZA	simulation of a dialog with a psychotherapist
Harper et al.	1969	English		random paragraph generation
Friedman	1969	English		testbed for transformational grammar
Winograd	1972	English	SHRDLU	description of a micro-world (cubes, blocks)
Simmons & Slocum	1972	English		sentence generation by using an ATN
Carbonell et al.	1973	English	SCHOLAR	natural language interface for teaching geography
Wong	1975	English		sentence generation from a semantic network
Goldman	1975	English	BABEL	paraphrase generation & lexical choice on the basis of abstract representations
Clippinger	1977	English	ERMA	simulation of performance errors (dialog with a psychiatrist)
Cohen	1978	English	OSCAR	speech act planning and planning of the message
Davey	1978	English	PROTEUS	generation of comments when playing TIC-TAC-TOE

Lehnert	1979	English	QUALM	goal sensitive elaboration of a response
Mc Donald	1980	English	MUMBLE	surface generator
Meehan	1980	English	TALESPIN	planning and generation of fables
Weiner	1980	English	BLAH	generation of explanations concerning income taxes
Carroll	1980	English		CALL system for teaching English
Moghrabi	1980	French		generation of cooking recipes
Bates & Ingria	1981	English	ILIAD	CALL system for transformational grammar
Mann & Moore	1981	English	KDS	generation of paragraphs
Kempen & Hoenkamp	1982	Dutch	IPG	incremental sentence product.
Mann & Matthiessen	1983	English	PENMAN	systemics sentence generator
Kukich	1983	English	ANA	generation of stock market reports
Swartout	1983	English	XPLAIN	explanation of the results of an expert system
Granville	1983	English	PAUL	generation of coherent texts
Wahlster et al.	1983	German	HAM-ANS	determination of the adequate level of a response
Mauldin	1984	English	KAFKA	sentence generation from a semantic network
Sigurd	1984	Swedish	COMMENTATOR	generation of scene descriptions
Buchberger & Horacek	1984	German	VIE-GEN	sentence generation from a semantic network
Appelt	1985	English	KAMP	modeling of the user's beliefs and goals
McKeown	1985	English	TEXT	use of schemata for determining content & text structure
Danlos	1985	French		use of discourse grammar
Simonin	1985	French		text plan optimization
Contant	1985	French	FRANA	French version of ANA
McCoy	1985	English	ROMPER	correction and anticipation of wrong inferences
Boyer et al.	1985	French		generation of paraphrases
Rösner	1986	German	SEMSYN	summaries about the labor market
Pollack	1986	English	SPIRIT	determination of the adequacy of information
Kehl	1986	German	GEOTEXT	description of geometrical configurations
Kronfeld	1986	English	BERTRAND	generation of definite descriptions

Patten	1986	English	SLANG	use of systemic grammars for NLG
Kittredge et al.	1986	English	RAREAS	generation of weather forecast reports
Novak	1987	German	NAOS	generation of coherent scene descriptions
Conklin	1987	English	GENARO	description of pictures
Jacobs	1987	English	KING	help system for UNIX
Kukich	1987	English		connectionist-based sentence generation
Jameson	1987	English	IMP	tailoring content by taking the user into account
de Finney et al.	1987	French	PARDA	generation of arguments
Mellish	1987	English		text generation from plans
André et al.	1987	German	SOCCER	generation of soccer reports
Nirenburg	1987	English	DIOGENES	generation for machine-translation
Ishizaki	1988	Japanese		generation of reports about terrorist crimes
Busemann	1988	German	SUTRA	surface generator of HAM-ANS dialog system
Houghton & Pearson	1988	English	DORIS	planning dialogues (vocal output)
Gabriel	1988	English	YH	simulation of conscious processes of writing
Jacobs	1988	English	PHRED	multilingual generator
Hovy	1988	English	PAULINE	pragmatically motivated text planning
Paris	1988	English	TAILOR	tailoring the text to the user's knowledge
Iordanskaja et al.	1988	English	GOSSIP	implementation of the meaning-text theory, paraphrase generation
Miller & Rennels	1988	English	PROSENET TEXNET	generation of clinical summaries
Finkler et al.	1989	German	POPEL-HOW	incremental sentence generation
Pemberton	1989	English	GESTER	story generation
Maybury	1989	English	GENNY	use of RST relations for producing coherent texts
Jablonski et al.	1990	German	NUGGET	text generation from a linguistic basis
Bourbeau et al.	1990	English French	FOG	mutlilingual generation of weather forecast reports
Gailly	1990	French	HERMES	processing of quantifier scoping
Cawsey	1990	English	EDGE	plan-based generation of explanatory discourse

Dale	1990	English	EPICURE	generation of cooking recipes
de Smedt	1990	Dutch	IPF	parallel-incremental sentence generation
Meteer	1990	English	SPOKESMAN	producing texts by revising
Mitkov	1990	Bulgarian	GECO	production of geometrical descriptions
Fawcett et al.	1990	English	GENESYS	with NIGEL the largest grammar of English
McKeown et al.	1990	English	COMET	generation of explanations (multimodal output)
Kreyss & Novak	1990	German	PIT	text planning
Namer	1990	French		generation of summaries of Verdi's operas
Reithinger	1990	German	POPEL-WHAT	incremental sentence generation
Moore et al.	1990	English	EES	generation of explanations
Zukerman	1991	English	WISHFUL	content planning for teaching algebra
Nogier	1991	French	SYLAE	sentence generation from conceptual graphs
Zock	1991	French	SWIM	CALL system for learning to speak French
Patten et al.	1992	English	ANITA	NL interface for a travel agent
Ward	1992	English	FIG	connectionist generator
Smadja & McKeown	1992	English	COOK	use of collocation for generating stock market reports
Kantrowitz & Bates	1992	English	GLINDA	narration and intercharacter communication
Rubinoff	1992	English	IGEN	Weather forecast generation
van der Linden	1992	English	IMAGENE	instructions for operating telephones
Fawcett et al.	1992	English	COMMUNAL	generation with systemic functional grammar
Roesner & Stede	1992	German	TECHDOC	multilingual instructions for automobile maintenance
Reiter et al	1992	English	IDAS	generation of on-line help documentation
Inui et al.	1992	Japanese	WEIVER	revision-based architecture
Elhadad	1992	English	FUF	implementation of Functional Unification Grammar
Carenini et al.	1993	Italian	ALFRESCO	NL interface to a videodisc of frescoes and monuments
Robin	1993	English	STREAK	revision-based generation of summaries
Wahlster et al.	1993	German	WIP	multimodal generation, instructions for technical products
McKeown et al.	1994	English	PLANDOC	summary generation

Cadlwell & Korelsky	1994	English	EXCLASS	generation of job descriptions
Cline	1994	English	KALOS	Knowledge-based generation with revision
Jokinen	1994	English	CDM	generation of explanatory dialogues
O'Donnell	1994	English	WAG	systemics sentence generator
Huang	1994	English	PROVERB	natural language proofs
Scott, Paris et al.	1995	E/Ger/It	DRAFTER	multilingual instructions
André	1995	German	PREPLAN	plan-based multimedia generation
Bateman & Teich	1995	English	KOMET	multilingual generation with systemic functional grammar
Buchanan et al.	1995	English	MIGRAINE	reactive patient explanations
Cawsey et al.	1995	English	PIGLIT	personalised patient explanation
Gagnon & Lapalme	1995	French	PRETEXTE	generation of temporal expressions
Kosseim	1995	French	SPIN	planning of instructional texts
Nicolov, Mellish & Ritchie	1995	English	PROTECTOR	approximate generation from conceptual graphs
Scott et al.	1996	E/Ger/It	GIST	**G**enerating **InS**tructional **T**ext
Coch	1996	Fr/E/Sp	AlethGen	multi-paragraph toolbox
Fasciano & Lapalme	1996	English	PostGraphe	generation of statistical graphics and text
Milosavljevic & Dale	1996	English	PEBA-II	encyclopedia descriptions
Gromova, Mellish & & Nicolov	1996	Russian	GeneRus	classification-based generation in Russian

Index